UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

Titulació:

Bachelor's degree in aerospace vehicle engineering

Alumne:

Lidia Carós Roca

Títol TFG:

Study of aerodynamic geometry optimization with RMOP2.0 and OpenFOAM

Director del TFG:

Martí Coma Company

Codirectors:

Josep Maria Bergadà Granyó

Jordi Pons Prats

Convocatòria de lliurament del TFG:

June 2019

Contingut d'aquest volum:

Report

Study of aerodynamic geometry optimization with RMOP2.0 and OpenFOAM

Report

Lidia Carós Roca

Director:
Martí Coma Company
Codirectors:
Josep Maria Bergadà Granyó
Jordi Pons Prats

Final Degree Thesis

Bachelor's degree in aerospace vehicle engineering

Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa
Universitat Politècnica de Catalunya

June 2019

Acknowledgement

Many contributions have been made to this study during its development. I want to thank my tutors Martí Coma Company, Jordi Pons Prats and Josep Maria Bergadà Granyó for their support and contribution in the knowledge acquired. I would like to thank Josep Maria Bergadà Granyó for his constant support during the realisation of the Thesis, being always available to listen and share his deep knowledge in Computational Fluid Dynamics, he has been of great help. On the other side, I would like to express my gratitude both to Martí Coma and Jordi Pons for the opportunity they have given me, being able to use the innovative RMOP optimization platform for my Final Bachelor Thesis and being able to compute CFD cases with the cluster at the research centre they belong, CIMNE.

Abstract

The focus of this Final Bachelor Thesis is the coupling and automation of a CFD case with an optimizer in order to produce an autonomous optimization process for an aerodynamic geometry. The case that will be used as test of the coupling will be the position and deflection optimization of a Fowler flap. The Thesis is composed of 8 chapters. In the *Chapter 1*, a brief introduction about the Open Source CFD Software *OpenFOAM* and the Robust Multi-Objective Optimization Platform *RMOP2.0* is made to be able to follow along the study, ending with the state of the art of optimization procedures for aerodynamic purposes. In *Chapter 2* the case that is being analysed and optimized is defined, its geometry and physical properties, also the resolution procedure is presented. The pre-process needed before solving the case composed by the geometry, domain, mesh creation and definition of the boundary conditions, is discussed in *Chapter 3*, together with its automated process. The *Chapter 4* consists in the selection of how the CFD will be solved and which turbulent model will be used, discussing the most suitable options considering both the case requirements and the computational resources and time available. Afterwards, on *Chapter 5* the OpenFOAM case is set up according to what has been decided, and the solving process is automated. The *Chapter 6* is based on the preparation of the optimizer for the coupling, describing all the parameters that have to be set and the essential files for the automated optimization process. Finally, the output results of the optimization test case will be exposed and discussed on *Chapter 7*. The main conclusions of the Thesis are presented in *Chapter 8* along with the improvements and future work of the study.

The chapters will be followed by the Appendixes and the References of the study.

I declare that,
the work in this Degree Thesis is completely my own work,
no part of this Degree Thesis is taken from other people's work without giving them credit,
all references have been clearly cited,
I'm authorised to make use of the research group related information I'm providing in this document.
I understand that an infringement of this declaration leaves me subject to the foreseen disciplinary actions by The Universitat Politècnica de Catalunya–BarcelonaTECH.

Lidia Carós Roca		10/06/2019
Student name	Signature	Date

Title of the Thesis: Study of aerodynamic geometry optimization with RMOP2.0 and OpenFOAM

Contents

List of Figures

List of Tables

Aim

The aim of the study is to achieve the automated coupling of an optimization algorithm with the Computational Fluid Dynamics process to solve the fluid around an aerodynamic geometry, in order to optimize its response. The coupling will be done between the innovative optimizer RMOP2.0, a CIMNE multi-objective platform based on genetic algorithms and evolutionary strategies, the recognized open source CFD software OpenFOAM, and the pre-process open source platform Salome. The geometry that will be used as test of the coupling is composed by a 2D airfoil and a Fowler flap, where the position and the deflection angle of the flap are the changing variables. The pre-process, solving, and post-process of the case will have to be automated with the capacity to adapt to the geometry changes.

Scope

As the present Thesis uses Computational Fluid Dynamics for solving the optimization case, it is important to comprehend the concepts about its functioning. Moreover, since OpenFOAM is used as the solver and gives the user the opportunity to choose the solving specifications and turbulence models, the theory behind this matters will be learnt and understood. Also, the manipulation of OpenFOAM will have to be thoroughly learnt in order to be able to automate its process.

The other main matter of the optimization coupling is the optimizer itself, RMOP2.0. The strategies that RMOP2.0 uses to optimize, multi-optimization and genetic algorithms, will be defined and comprehended in order to later specify the inputs of the optimization and success in the coupling.

To be able to optimize a case, the sub-processes coupled with the optimizer (pre-process, solver and post-process) are needed to be automated. For this reason, the creation of domain and mesh will have to be done with a platform that enables code entries a part from graphic interface. The selected software has been Salome9.2.1 due to its Python entry. Both graphic interface and Python usage will have to be assimilated.

A study will have to be made in order to asses if the optimization is reliable or not. This will be done by the capacity of different strategies of comparison between the different flap positions. However, a high number of suppositions will have to be made in order to complete the optimization process in the available time for the Thesis.

Intermediate files will have to be created in order to manage the data between the optimizer and the CFD process. In order to do so, programming skills in different programing languages will have to be acquired.

Finally, a 2D configuration of airfoil with flap will be optimized, the results of which will be discussed and future work will be proposed based on the aspects that can be improved.

Requirements

The main requirement of the Thesis is to obtain an optimization process for an aerodynamic geometry that finds the better solutions in an autonomous way, without intervention. The automated process has to be done coupling the optimizer RMOP2.0 and OpenFOAM as the CFD software.

The autonomous process must have been designed with an exit when exceeding a determined time or a set number of iterations, avoiding an endless loop. Besides, it is needed to set values of the solution far from the normal and optimum values if the process breaks at some point. This way the optimizer will not try to return to that solution.

In order to achieve that the automated process functions correctly it is required that the mesh and the boundary conditions adapt properly to the geometric changes.

It is also mandatory that the CFD mesh has acceptable results to compare the different options in order to find the optimum one. The case will have to be properly defined according to its physical properties. The turbulence model and the solver used will have to be according to the case needs and the available resources.

The computational time has to be sufficiently low to be able to run a number of evaluations that allow the evolution of the solution, all this within the Thesis available time.

The computational time required for this study is the reason why no accuracy requirements of the solution values are demanded. As the process can be used for multiple cases adapting the mesh and the solver, higher accuracy can be achieved when the time permits it.

Justification

In aerospace engineering, due to the demand of high performance, the modelling and optimization tools are emerging. Optimization algorithms allow improving the previous designs without testing a high number of possibilities until finding the optimum.

The International Centre for Numerical Methods in Engineering, CIMNE, is a research organization which develops numerical methods and computational techniques for advancing knowledge and technology in engineering and applied sciences. The researchers of CIMNE have developed RMOP2.0, a Robust Multi-Objective and Multidisciplinary Optimization Platform, which can be coupled with other software in order to optimize a determined situation.

Coupling RMOP2.0 with a recognised CFD solver such as OpenFOAM can provide a high increase in the results of an aerodynamic case, optimizating geometries or parameters that would require much more time for human capacities to obtain the same optimum results.

This study is a preliminary coupling of OpenFOAM and RMOP2.0 in order to optimize 2D geometries, that can be later extrapolated to more complex studies.

Chapter 1

Introduction

A background is needed in order to understand all the concepts of the present Thesis. In this Chaper, an introduction of the platforms used in this work is made to comprehend the functioning of both OpenFOAM and the *Robust Multi-Objective Optimization Platform* (RMOP2.0). Afterwards, a state of the art about the optimization of aerodynamic geometries is addressed.

The explanation about OpenFOAM starts with an introduction about Computational Fluid Dynamics (CFD), after, a brief description of the OpenFOAM use of solution algorithms is made and finally, the case folder configuration needed to solve a determined case is presented.

RMOP2.0 is introduced together with the description of the main optimization parameters, the objective functions and design variables. Afterwards, an overview of its use of multi-objective optimization and genetic algorithms is exposed.

1.1 OpenFOAM

OpenFOAM, *Open source Field Operation And Manipulation* is an open source C++ toolbox that develops numerical solvers for Computational Fluid Dynamics.

1.1.1 Computational Fluid Dynamics

In order to solve the fluid around a geometry OpenFOAM uses the finite volume method following three main steps: the integration of the governing equations of fluid flow over all the control volumes of the domain, the discretisation to have a system of algebraic equations and the solution of this equations by an iterative method.

1.1.1.1 Governing equations

The governing equations of fluid flow represent mathematical statements of the conservation laws of physics. The fluid will be regarded as continuum and the equations will be stated for a Newtonian fluid. [2]

Considering an element of fluid with sides δx, δy and δz, the governing equations on the fluid element are:

– **Mass conservation**

The mass of the fluid is conserved, thus, the rate of increase of mass in the fluid element equals the net rate of flow of mass into the fluid element.

$$\frac{\delta \rho}{\delta t} + div(\rho u) = 0 \tag{1.1}$$

– **Momentum equation**

Newton's second law states that the rate of increase of momentum of the fluid particle equals the sum of the forces on the particle.

$$\frac{\delta(\rho v)}{\delta t} + div(\rho v u) = -\frac{\delta p}{\delta x} + div(\mu \, grad\, v) + S_{M_X} \tag{1.2a}$$

$$\frac{\delta(\rho v)}{\delta t} + div(\rho v u) = -\frac{\delta p}{\delta x} + div(\mu \, grad\, v) + S_{M_Y} \tag{1.2b}$$

$$\frac{\delta(\rho w)}{\delta t} + div(\rho w u) = -\frac{\delta p}{\delta x} + div(\mu \, grad\, w) + S_{M_Z} \tag{1.2c}$$

– **Energy equation**

The energy equation comes from the first law of thermodynamics, and it states that the rate of increase of energy in the fluid particle equals the sum of the net rate of the heat added to the fluid particle and the net rate of work done on the fluid particle.

$$\frac{\delta(\rho i)}{\delta t} + div(\rho i u) = -p\, div\, u + div(k\, grad T) + \Phi + S_i \tag{1.3}$$

1.1.1.2 Solution algorithms

As the pressure gradient is not known beforehand, it is not direct to obtain the discretised equations for velocities from the momentum equations. Hence, an iterative algorithm must be applied in order to solve the governing equations. These algorithms (SIMPLE and PISO) are explained in depth in the solver section 4.2.

Regarding the solution of the system of linear algebraic equations, it can be solved with direct methods or iterative methods, some examples are TDMA, Jacobi, Gauss-Seidel. The resolution of these algorithms will not be detailed on the present Thesis but information about it can be found in (H.K. Versteed and W.Malalasekera, 2007) [2].

1.1.2 Usage of OpenFOAM

In order to solve a determined case with OpenFOAM, it is essential to prepare the case according to the simulation that will be run. OpenFOAM allows the user to have a complete control in all the parameters that contribute in the solving of a case, this matter makes the user understand deeply the functioning of the CFD process.

There are files and directories that are mandatory to create a case. A brief introduction will be made about the OpenFOAM case and its composing folders and files. Further explanation of these files and properties chosen are thoroughly explained in the *OpenFOAM case* Chapter 5. However, an introduction about the files is necessary in order to follow along the Thesis.

The main folder of any case that is going to be solved with OpenFOAM needs to contain three folders: the time directory, the constant folder and the system folder.

The time directory contains the initial and boundary conditions for the case. These conditions have to be defined for the pressure field and the velocity field, in order to be able to solve the discretized governing equations. Also the initial and boundary conditions of the turbulence parameters have to be add, but the files required depend on the turbulence model used.

The constant directory contains all the information about the mesh in the polyMesh folder. In the directory are also placed the files containing the turbulence (turbulence model) and transport (density, viscosity, etc) properties.

The system directory contains the group of files that control the solving. Three files are essential inside this folder, and, in case the solving is wanted to be parallelised the decomposeParDict will also be necessary. The controlDict file controls the time or evaluations of the solution and the writing of the outputs. The fvSchemes file controls the numerical schemes used in the simulation and the fvSolution controls the solution algorithms and the tolerances that have to be achieved.

The case structure mentioned, with its containing folders and files, is represented on the Figure 1.1.

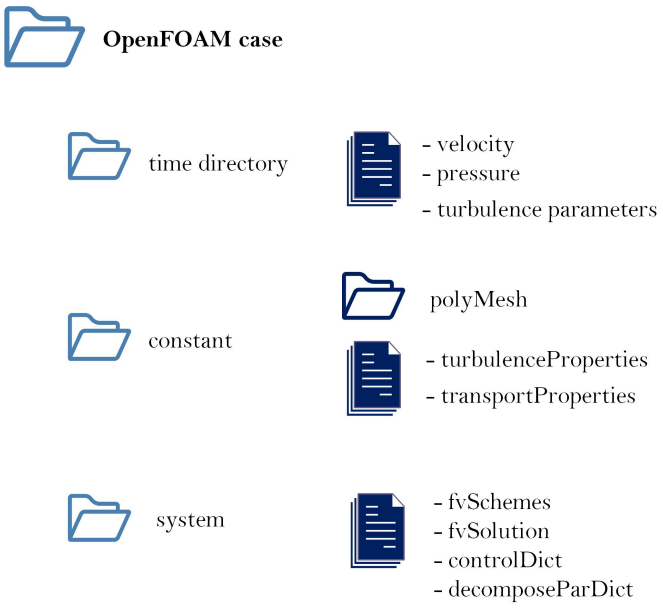


FIGURE 1.1: Case structure

1.2 RMOP2.0

The *Robust Multi-Objective Optimization Platform*, RMOP is a multi-objective optimization platform based on genetic algorithms and evolutionary strategies, developed by CIMNE (*International Centre for Numerical Methods in Engineering*). It allows to find optimal solutions for problems with multiple objective functions and design variables and can be easily coupled with an autonomous process. [3]

1.2.1 Objective functions and design variables

The objective function is the best solution or group of solutions that the model aims to find, this objective function is either to be minimized or maximized. The design variables are the changing inputs to the process that change the solution of the objective functions. The better the resulting objective function, the most value has that design variable.

1.2.2 Optimization problem

A problem of optimization through minimization can be represented with the following formulation:

Given a function $f : R^n \to R$

Search an element x_0 in R^n such that $f(x_0) \leq f(x)$ for all x in R^n

where f is the objective function to be minimized and R^n is a set of real numbers.

1.2.3 Multi-objective optimization

Involving more than one objective function to be optimized simultaneously implies a multi-objective optimization. In mathematical terms, a multi-objective optimization problem can be formulated as stated in [4]:

Given m objective functions $f_1 : \chi \to R, ..., f_m : \chi \to R$

which map a decision space χ into R, a multi-objective optimization problem (MOP) is given by the following problem statement

$$minimize(f_1(x), f_2(x), ..., f_m(x)) \qquad \text{x} \in \chi$$

where the integer m > 1 is the number of objective functions.

In the multi-objective optimization there is no solution that optimizes each objective at the same time.

The highest valued solutions form the Pareto Front, a set of equally good solutions, since its values for the objective functions can not be improved without degrading some of the other objective values. These are also called non-dominated or non-inferior solutions, as they dominate all the other solutions except the ones in the Pareto.

A representation of the Pareto is shown on the Figure 1.2.

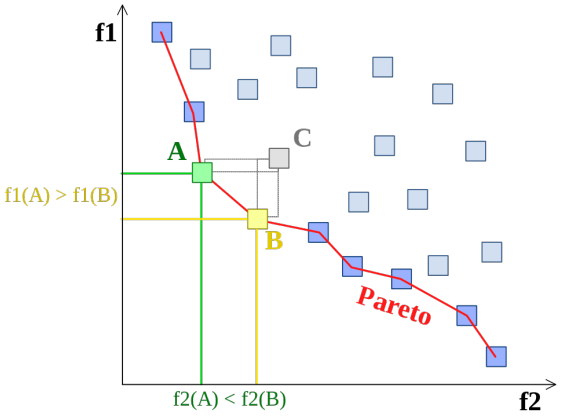


FIGURE 1.2: Pareto Front [1]

1.2.4 Genetic Algorithms and Evolutionary Strategies

Genetic algorithms are solution-search or optimization techniques inspired by Charles Darwin's theory of natural evolution.

The algorithm reflects the process of natural selection where the fittest individuals, who can adapt to changes in their environment, are able to survive and reproduce to next generations.

The GA problem operates on a population of individuals, each individual is characterised by a set of parameters (design variables).

Every individual has a solution to the problem that is valued through the objective functions obtained by that individual, these values measure the fitness of the individual, how good it is to the particular problem.

1.2.5 Optimization process

The optimization process starts with a random population of individuals and carries out a process of fitness-based selection to designate the most valuable individuals. These individuals are selected as *parents*, and their properties are recombined to produce a successor generation of *children* that has more valuable results.

As this process is iterated, the successive generations of individuals evolve and the average fitness of the design variables tends to increase until some stopping criterion is reached.

To prevent the algorithm to be blocked in a local minimum, some of the children individuals formed are subjected to a mutation with a low random probability.

1.2.6 Coupling with an automated process

The optimizer RMOP2.0 outputs the design variables that have to be tested and reads the objective functions obtained for these design variables in order to evaluate them and produce new better generations.

The process between the given design variables and the obtainment of its objective functions is the task that the *analyser* has to carry out. The analyser is all the processes needed to be done in an automated way to evaluate the objective functions for a set of design variables.

The current Thesis works as the analyser of the optimization, as it has to be automated the process from the design variables definition, going through the pre-process and solving of these variables, to the output of the CFD results as objective functions.

1.3 State of the art

New developments are being accomplished in a wide range of fields thanks to optimization algorithms. If an optimisation process is coupled with a Computational Fluid Dynamics resolution, new improved aerospace features can be optimized in order to obtain a better performance. Consequently having an optimum result which, in the case of having multiple design variables and objective functions, is out of the range of human capacities.

Multi-objective optimizations are currently being applied to aircraft wing design, using Computational Fluid Dynamics in order to achieve optimum results without a costly and large testing process.

These otpimizations are mainly focused in shape changes in aerodynamic geometries. Some examples of these are the variation of wing surface and airfoil shapes in order to reduce the fuel consumption [5] or the aim to minimize the total drag of an aircraft modifying the geometry [6].

This kind of approach is becoming essential for companies in the aerospace field due to the demanding high performance, and, as aerospace systems require the coupling of the multiple disciplines in order to obtain a successful design, multidisciplinary design optimizations (MDO) are being developed. These multiple disciplines contemplate aerodynamics, structures, propulsion and control mainly, and these are highly connected between them sometimes in non intuitive ways. In these types of optimization, the design variables consider the different disciplines.

One of the first applications of multidisciplinary design optimization was made on wing design, coupling the aerodynamics and structures for flexible wing structures subject to strength and induced drag constraints [7]. This has been later assessed by many other authors in different aerostructural optimizations such as [8] or [9].

With the high advancements made in computational power, automated design optimisation procedures have become more competent and are currently being developed in a wide range of fields. Also, the optimization algorithms have been improved substantially in the past decades. However, difficulties are found in aerodynamic optimizations to define methods capable of operating many iterations within a realistic run time, but sophisticated enough to capture the information required. Moreover, in multidisciplinary optimizations, adversity is found when modelling the complex connections between the disciplines that reign the aerospace systems.

Chapter 2

Optimization case description

The aerodynamic case that is being optimized is explained on this chapter, with its geometric and physical parameters. In addition, the optimizing process is schematised with a description of its composing stages.

2.1 Aerodynamic geometry

The geometry that is going to be optimized is composed by an airfoil and a Fowler flap. The optimization will be based in finding the best flap position changing its coordinates (dx and dy) and deflection angle (δ), acting as the design variables, in order to obtain the highest lift coefficient and efficiency, the objective functions.

The initial geometry is a NASA/LANGLEY LS(1)-0417 GA(W)-1 *General Aviation Airfoil* with a Fowler flap at the 29% of the chord, a commonly used composition. The geometry is shown in the Figure 2.1.

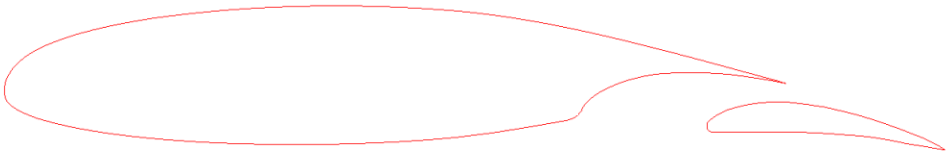


FIGURE 2.1: GA(W)-1 Airfoil with a 29% chord Fowler flap.

The airfoil that will be computed has a chord of 0.1m and the Fowler flap of 0.01. The flap movements during the optimization will be made around the downstream region of the airfoil without entering no-sense positions, its angles of deflection will go from 0 degrees to 30.

The deflection range could be larger but the interesting points of the optimization are the ones that achieve optimum results both in efficiency and lift coefficient. Furthermore, as will be seen in the meshing chapter, having a wide range would worsen the mesh in some positions and maybe produce invalid results.

The airfoil will be analyzed with an angle of attack of 4º as it has a good aerodynamic performance while the boundary layer detaches smoothly, this way, turbulence models can be applied with less error.

2.2 Physical properties of the case

In order to avoid a high computational time in solving, a low Reynolds number is needed, since the size of the mesh cells and the computational domain dimensions depend directly on it. The Reynolds number is defined as follows.

$$Re = \frac{\rho v_\infty c}{\mu} \tag{2.1}$$

where ρ is the air density,

v_∞ is the velocity of the upstream flow,

c is the airfoil chord,

μ is the dynamic viscosity.

The upstream flow velocity at which the case is computed is $v_\infty = 5m/s$. Having a chord of $c = 0.1m$, an air density of $\rho = 1.225kg/m^3$ and a dynamic viscosity of $\mu = 1.81 \times 10^{-5}$ kg/ms, it results a Reynolds number of:

$$Re = \frac{1.225 \times 5 \times 0.1}{1.81 \times 10^{-5}} = 3.4 \times 10^4 \tag{2.2}$$

Also, a low Mach number is obtained as the velocity is low.

$$M = \frac{v}{c} = \frac{v}{\sqrt{\gamma RT}} \tag{2.3}$$

where v is upstream flow velocity,

c is the speed of sound in the fluid at the flow temperature,

γ is the specific heat ratio,

R is specific gas constant $[J/kgK]$,

T is the temperature of the flow $[K]$.

$$M = \frac{5}{\sqrt{1.4 \times 287.058 \times 288.15}} = 0.0147 \tag{2.4}$$

As M«0.3, the flow can be treated as incompressible.

2.3 Procedure of resolution

The optimizer changes the flap positions (dx, dy and deflection angle δ), the case for each flap position is prepared and solved and the results of this position are given to the optimizer, which evaluates them and produces new better positions.

2.3.1 Salome

For the computational domain creation around the airfoil and the design of the mesh it has been used Salome 9.2.1. This open source software has been chosen due to its Python interface, which has access to all its functionalities via scripts. Thanks to it, the meshing process can be automated. While the position and angle of the Fowler flap change, the computational domain remains constant and the mesh is adapted to the new geometry. This mesh is later exported and converted into OpenFOAM format.

2.3.2 OpenFOAM

The mesh file will be replaced each time into the folder of the OpenFOAM case, in particular inside the polyMesh folder. The rest of the folders and files of the case will remain constant in all individuals, as the physical conditions of the case and the solving parameters are the same.

2.3.3 Post-Process

The CFD solution calculated is placed in a solution folder created by OpenFOAM, the values of the aerodynamic coefficients are selected and written in the *Eval.individual* file so the optimizer can evaluate this individual and re-write the design variables to achieve better results.

2.3.4 RMOP2.0

In order to accomplish the optimization *RMOP2.0* uses three essential files. These are the following: a unique script that will automatically run the desired case called *analyser*, a file for the design variables (*Eval.DVs*), which will be written by the optimizer, and the values of the objective functions (Cl and E) of the current individual (*Eval.individual*), that will be written after the solving by the *analyser*.

The procedure that will be followed to couple the optimization with the sub-processes is represented in a simplified flow diagram on the Figure 2.2. A more detailed flow diagram will be shown on the coupling Chapter 6, with all the files needed for the optimization process.

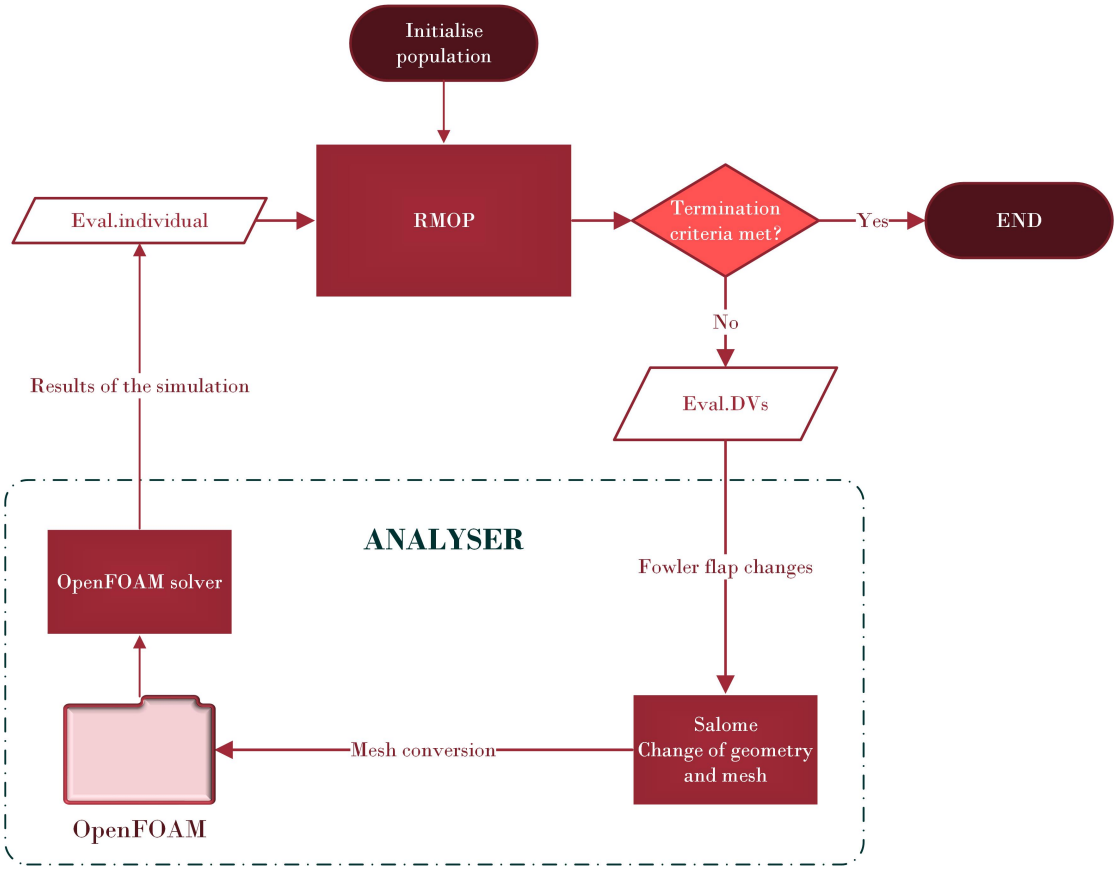


FIGURE 2.2: Flow diagram of the procedure of resolution.

Chapter 3

Pre-process

Before any type of optimization can occur, the execution case needs to be prepared and automated for each individual. This Chapter consists in the explanation of the pre-process followed every time the flap position changes. This pre-process contemplates different steps: update of the geometry and computational domain, re-meshing, and re-assigning the boundary conditions.

First, the actualization of the coordinates of the flap has to be carried out according to the optimizer proposed variations. Afterwards, the computational domain needs to be created around this points, with the different sections corresponding to the different mesh densities. Subsequently, the mesh is updated and validated for the new position, and it is exported and converted for the OpenFOAM case.

As said previously, the geometry, domain and meshing part of the case is made with Salome 9.2.1 due to its integrated Python console. All the Salome scripts can be found in the Appendix A.1.

3.1 Geometry

The geometry of the airfoil and flap is constructed in Salome. First, it is necessary to import the points that compose the initial airfoil and flap to the program. A sufficient number of points to define the airfoil and flap are needed in order to create a continuous geometry for the CFD process. Afterwards, a B-Spline interpolation line has been created going through all the points in a smooth way. After having the geometry, it is needed not only to automate the explained process but also to add the flap changes (dx, dy and angle deflection δ) to this geometry.

3.1.1 Geometry automation

To start the automated process, a Python script (*GEOMmodification.py*) is needed to add the variations (dx, dy and angle deflection δ) to the flap coordinates. This script writes both flap and airfoil coordinates in another Python script (*POINTS.py*) in Salome commands to import them afterwards into the Salome study.

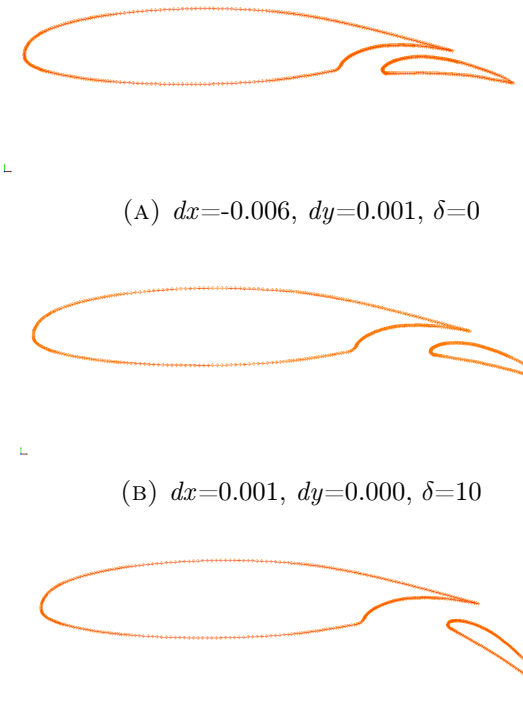
```
f=open( POINTS.py , wa )
... # writing the necessary salome commands
for i in range(len(xPtsF)): # add translation changes on X and Y
   f.write( fowler_Pt_%s %(i) +  = geompy.MakeVertex( +str(xPtsF[i]+dx)+ , +str(↩
       yPtsF[i]+dy)+ ,0) + \n )
# create reference point and vector for rotation
f.write( Vertex_1 = geompy.MakeVertexWithRef(fowler_Pt_97 , 0, 0, 0.1) + \n )
f.write( Vector_1 = geompy.MakeVector(fowler_Pt_97 , Vertex_1) + \n )
for i in range(len(xPtsF)): # make rotation
   f.write( Fowler_Pt_%s %(i) +  =  geompy.MakeRotation(fowler_Pt_%s %(i) + , ↩
       Vector_1 , - +str(angle)+ *math.pi/180.0) + \n )
...
f.close()
```

Listing 3.1: GEOMmodification.py

As can be seen on the Listing 3.1, While the addition of the flap points into the study (`fowler_Pt = geompy.MakeVertex`), the dx and dx variations are added into the coordinates. In order to add the deflection change, it is needed a Salome command which will deflect the flap after the coordinates are written (`Fowler_Pt = geompy.MakeRotation`) referenced with a vertex in the z direction.

The airfoil coordinates are also added in this script for further developments of the project where the airfoil shape could also be modified.

Once the points are imported, interpolation lines passing through the points are created around the airfoil and flap to construct its geometry, this can be seen on the Figure 3.1.



(A) dx=-0.006, dy=0.001, δ=0

(B) dx=0.001, dy=0.000, δ=10

(C) dx=0.003, dy=0.001, δ=30

FIGURE 3.1: Automated flap positions changing dx, dy and the angle deflection δ.

As these lines are constructed by the point names (`Fowlerext1 = geompy.MakeInterpol ([Fowler_Pt_87, Fowler_Pt_93, Fowler_Pt_94, Fowler_Pt_95, ...])`), if the number of points that define the airfoil and flap is maintained constant, the lines will adapt automatically to the flap changes.

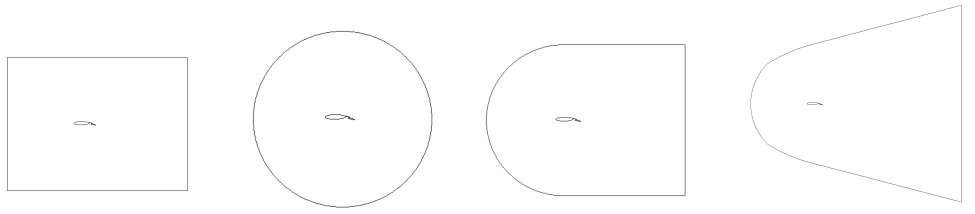
These lines are created by segments as the mesh is going to be made by parts to make sure all the flow behaviour is captured, this domain division for the mesh is explained in the Section 3.2.3.

3.2 Computational domain

The computational domain is the region around the airfoil to be evaluated where the numerical equations of fluid flow are solved by CFD. In order to do so, the domain is divided in cells, creating the mesh, and afterwards, the correspondent boundary conditions must be applied. The dimensions and shape of the domain must be carefully chosen in order to solve correctly the flow equations around the airfoil.

3.2.1 Shape

There are different types of computational domain shapes used to compute the fluid flow around an airfoil, squared domain, O-grid, C-grid... Depending on the purpose and detail of the CFD to solve it can be chosen one or another. Some domains can be wrongly chosen by not considering the effects of the angle of attack of the fluid flow in the inlet and outlet, some examples are represented in the Figure 3.2.



(A) Squared-grid (B) O-grid (C) C-grid (D) Domain chosen

FIGURE 3.2: Other computational domain shapes.

In the case of study, the domain shape that has been chosen is the one shown in the Figure 3.2d in order to have only one inlet and one outlet. It is composed by a parabola as inlet and a vertical line as the outlet. The inclination of the parabola must be based on the maximum angle of attack that is going to be evaluated, in order to prevent the fluid to get out of the domain by the inlet wall. As the angle of attack on the case will be 4º, a slope of 15º is correct, chosen by the domain dimensions. This slope can be seen on Figure 3.3.

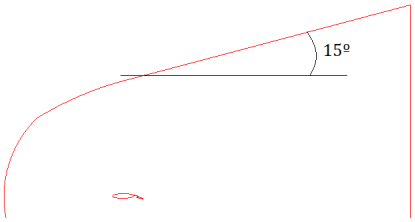


FIGURE 3.3: Computational domain angle.

3.2.2 Dimensions

The dimensions of the domain depend principally on the Reynolds Number of the case as different studies have proved. In this case, the dimensions have been chosen by similar and reliable Reynolds Number airfoil simulations such as the ones referenced in [2].

The resultant dimensions of the domain are the ones shown in the Figure 3.4. As can be seen, the upstream length is 5 chords from the airfoil leading edge, and in order to capture the airfoil effects in the downstream flow the outlet is 10 chords from the trailing edge. The vertical outlet line is 10 chords from the airfoil both upwards and downwards. The parabola is 8 chords in vertical distance from the leading edge point.

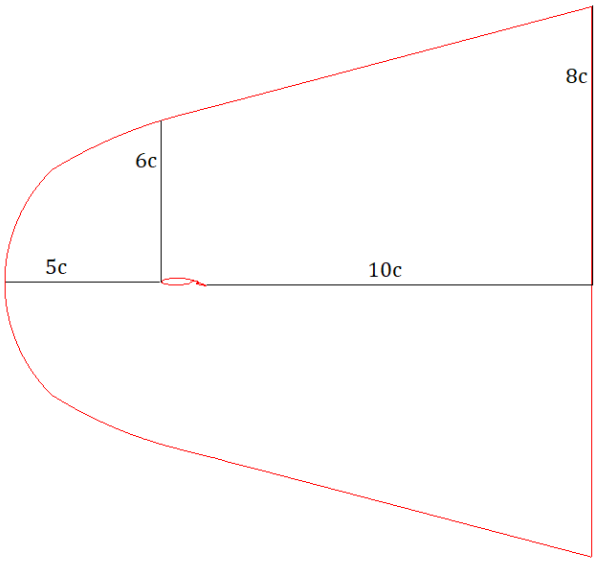


FIGURE 3.4: Computational domain dimensions.

3.2.3 Domain divisions

With the aim of creating a mesh capable of computing the fluid flow, the domain must be divided in different regions as the mesh will have different densities depending on its proximity to the airfoil and the perturbation of the flow. This different regions allow different discretisation of its edges. The increasing size of the mesh in far distances from the airfoil is made to reduce the computational cost on the regions that the fluid flow has low changing gradients on its physical properties.

3.2.3.1 Boundary layer domain

The boundary layer, where the wall surface is in contact with the stream flow, has a flow velocity variation from zero, at the wall, to upstream velocity (5m/s in this case) at the boundary. Its thickness can be considered to be within the 10% of the airfoil chord [10].

Inside the boundary layer, as in the downstream flow, is where the fluid flow has higher gradients on pressure and velocity, so it is where the mesh has to be more dense. Another critical region that has to be treated carefully is the one between the airfoil and flap. The resultant boundary domain is represented in the Figure 3.5.

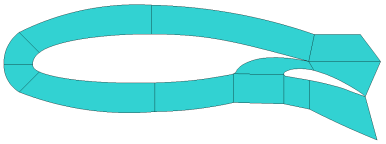


FIGURE 3.5: Boundary layer domain.

3.2.3.2 Outer domain

The outer domain has had to be created adapting it to the inner one, in order to make its meshes fit. Nevertheless, these regions are also necessary in order to achieve a continuous growth of the mesh without abrupt cell size changes. As can be seen in the Figure 3.6, more regions are needed in the downstream zone than in the upstream one as the mesh density has to be much larger to capture the wake. This downstream region needs to consider also the maximum angle of attack that will be computed, and thus, deduce the direction of the downstream flow. The number of regions on the upper and lower sides of the airfoil are not that important as the flow will be mostly undisturbed flow, these are basically determined by the boundary layer regions.

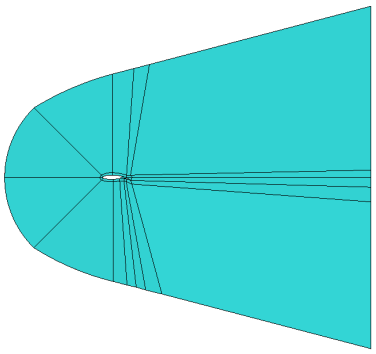


FIGURE 3.6: Computational domain.

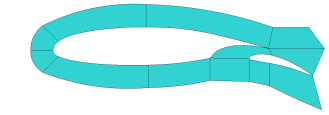
3.2.4 Domain automation

Lines and vectors must be created depending on the point names to change along with them when they change its coordinates, as it happened in the geometry automation. This dependencies must be chosen correctly as will highly influence the adaptive mesh to the flap variations.

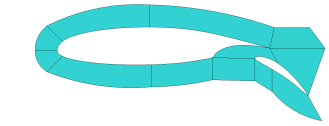
The automatic process of the domain creation starts firstly with the boundary layer domain, designing it in a way that wherever the Fowler flap moves, this regions will have a distance of 10%*chord* from the walls.

After creating this limiting boundary points, new ones are created in order to make arcs and lines to close the regions and create the faces. At first, some of the dependencies were made with the wrong reference points and when the flap position changed, the regions did not adapt to it, creating a distorted mesh.

After a trial and error procedure, the pertinent dependencies have been created and the result is the one demonstrated in the Figure 3.7. Regions having, for all the considered flap movements, similar domain sizes for its mesh to be correct.



(A)



(B)

FIGURE 3.7: Different automated domains around changing flap positions.

On the other hand, the outer domain distances do not depend on the changing points, as it is of no interest that its shape changes. In the case it did, it could distort the inlet parabolic line or the outlet straight vertical line. For this reason it has been assured that whatever the flap does, the outer domain lines will not vary. Of course, the lines and its respective faces connecting the boundary layer with the outer domain do depend on the points and do change, but this changes will only slightly increase or decrease the density of the regions near the flap, without altering the continuity of the mesh.

3.3 Mesh generation

Once having the domain created, its discretisation can begin. The discratisation is made on the region where the fluid will flow, leaving the solids (airfoil and flap) as void regions. The mesh must be created meticulously in order to achieve an accurate solution, a faster convergence and the reduction of numerical diffusion.

The generation of the mesh can be divided principally in structured and un-structured mesh. A structured mesh is the one having all nodes of the mesh an equal number of adjacent elements, except the ones in the edges. Even though the generation of a structured mesh has a higher difficulty, it has some remarkable aspects, being the reason why it has been chosen. Its better alignment with the flow imply better convergence as the grid lines follow the contours of the geometry just like the fluid does, whereas there's no such alignment in an unstructured mesh, also, it is less time consuming and memory requiring and it permits a higher control on its design, allowing a better action on its defining parameters.

As the geometry need to mesh has no abrupt irregularities it is going to be used an orthogonal curvilinear mesh. In an orthogonal mesh, the grid lines are intended to be perpendicular in their intersections. The mesh will start as two-dimensional, so the cells composing it will be quadrilaterals. These are going to be later extruded to hexahedras.

The resulting mesh is shown of the Figure 3.8, with a total of 28740 cells.

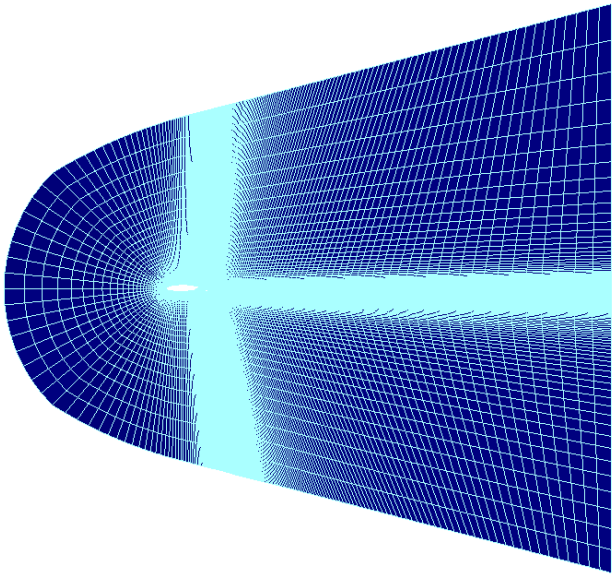


FIGURE 3.8: Resultant structured mesh.

3.3.1 Mesh parameters

The cell size will grow from the wall to the end of the domain, having smaller cells on the boundary layer where the viscous forces are the same order of magnitude or larger than inertial forces. The parameters of the mesh in this region, cell starting size and growth rate, will have to be carefully chosen .

3.3.1.1 Dimensionless wall distance (y^+)

When designing a mesh, in order to capture the fluid flow it has to be taken into account the dimensionless parameter related to the first cell height, y^+. Which is defined as

$$y^+ = \frac{\Delta y u_\tau}{\nu} \tag{3.1}$$

where Δy is the distance from the wall, u_τ is the friction velocity and ν is the kinematic viscosity.

When a fluid encounters a wall, two layers can be easily distinguished, the viscous wall region with $y^+ < 50$ where viscosity plays a major role, and the outer layer with $y^+ > 50$ where the effect of viscosity is negligible. However, the first layer can also be divided in sub-layers.

These regions are [11], [2]:

- **The linear or viscous sublayer ($\mathbf{y^+ < 5}$)**: The viscous effect dominates the fluid. It can be assumed that the total shear stress is approximately constant and equal to the wall shear stress, so the Reynolds shear stress is negligible. Thus, the *linear velocity law* is given by
$$y^+ = u^+ \tag{3.2}$$

- **The buffer layer ($\mathbf{5 < y^+ < 30}$)**: It is the transition region, viscous and turbulent stresses are of similar magnitude. The velocity profile is not well defined.

- **The logarithmic or log-law layer ($\mathbf{y^+ > 30}$)**: The shear stress varies very slowly with a logarithmic function along the distance from the wall. The relationship between u^+ and y^+ is
$$u^+ = \frac{1}{k} ln(y^+) + B \tag{3.3}$$
with the Von Karman constant $k \approx 0.4$ and the additive constant for smooth walls $B \approx 5.5$ (decreasing with higher roughness).

All things said, in order to capture the gradients in the viscous sublayer, y^+ must be $y^+ < 5$ and preferably $y^+ \leq 1$ [2]. However, the lower the y^+, the higher the computational cost. As this study is based in the optimization of the flap location, where the different positions are simulated and compared, as long as the results can be considered correct no further accuracy will be sought, as it would increase the computational time far above the available time for the Thesis. For this reason the selected y^+ has been $\mathbf{y^+ = 4}$.

The parameters that will be used during the creation of the mesh will be the first cell dimensions, height (Δy) and length (Δx).

3.3.1.2 First cell height (Δy)

Having decided the level of discretisation accuracy in the boundary layer (y^+), the Δy can be calculated with the relation mentioned in 3.3.1.1.

$$y^+ = \frac{\Delta y u_\tau}{\nu} \tag{3.4}$$

where

$$u_\tau = \sqrt{\frac{\tau}{\rho}} \qquad and \qquad \tau = \frac{C_f \rho V_\infty^2}{2} \tag{3.5}$$

where C_f can be estimated with the *flat plate approximation*

$$C_f = \frac{0.026}{Re^{1/7}} \tag{3.6}$$

Solving the equations with $y^+ = 4$, $\rho = 1.225$, $\mu = 1.81 \times 10^{-5}$, $V_\infty = 5m/s$ and $Re = 34000$, the resulting first cell height results $\mathbf{\Delta y = 5.1 \times 10^{-4}}$. On the Figure 3.9, the first cells can be observed.

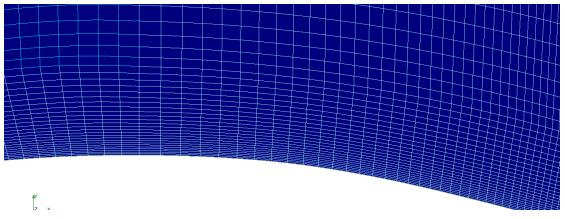


FIGURE 3.9: Boundary layer mesh.

The usage of the flat plate approximation means that the Δy selected and applied when meshing will not imply the intended $y+$, for this reason, when solving the CFD case the y+ will have to be calculated, and in the case that it is $y^+ > 5$, the geometry will have to be re-meshed.

3.3.1.3 Aspect ratio (Δx)

The Δx is not as important as the Δy since the highest gradients are encountered on the vertical direction, nevertheless, a high aspect ratio can miss the behaviour of the fluid. It is important to have an acceptable aspect ratio on the regions that the flow will be detached. As there will be significant changes both in y and x directions around the airfoil and flap, it has been used an aspect ratio below 1:10.

Special attention has been given to the region between and after the airfoil and flap, being crucial regions. The aspect ratio in this regions has been reduced, trying to achieve 1:1 aspect ratio.

3.3.1.4 Growth rate

As mentioned before, the boundary layer region needs a thinner mesh, thus, it will have a smaller growth rate than the outer region, this is why different surfaces are created when generating the domain.

In the boundary layer the mesh will have a 1% growth rate to capture the flow behaviour. The rest of the mesh, from the last cell on the boundary layer until the limits of the domain, will have a growth rate of 8%. This second percentage has been chosen knowing the first cell size and the distance that was needed to discretise, while the growth rate was intended to be around a 10%.

In the region between the airfoil and flap no growth rate has been applied, all the cells have the minimum Δy and, as said in the previous subsection, Δx. This can be clearly seen on the Figure 3.10.

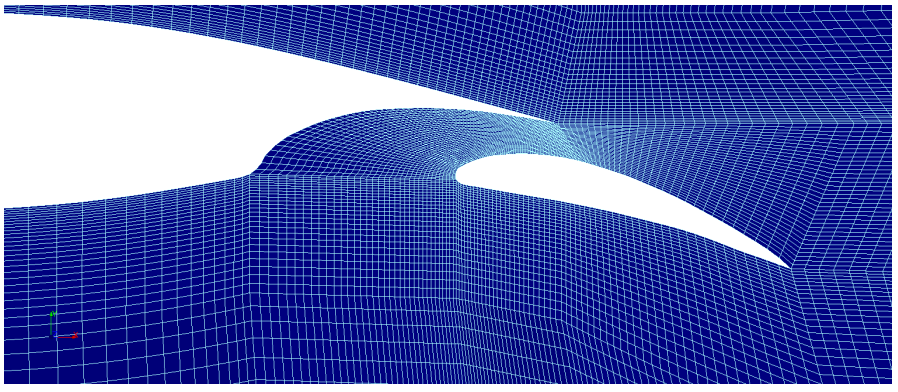


FIGURE 3.10: Mesh between the airfoil and flap.

The mesh on the downstream flow has a higher mesh density than the other outer domain regions, as can be seen on the Figure 3.11. Its growth in the x direction has been chosen paying attention to the aspect ratio, being under 1:40 on the region near the airfoil.

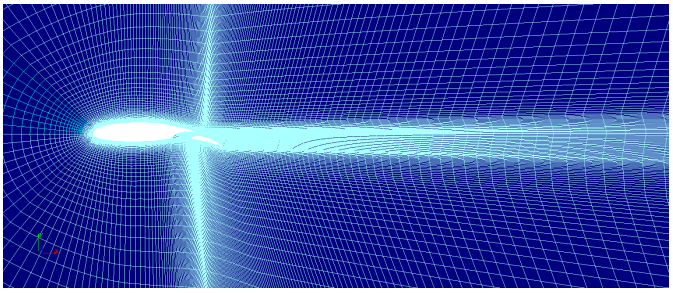


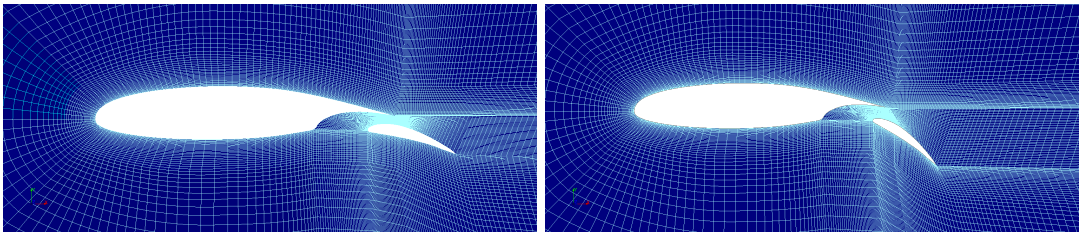
FIGURE 3.11: Mesh with smaller cells on the downstream region.

3.3.2 Mesh automation

For each geometry and domain generation, the mesh is created, depending directly on the surfaces mentioned in 3.2. The mesh is created discretising the edges of each surface, choosing the growth rate or the number of cells. After, all the meshes are computed. It is obviously of importance that the surface grids with shared edge have the same properties in order to merge correctly all the surface meshes.

As the flap movements occur, the mesh will change along with it, for this reason it must be designed wisely so there is no position chosen by the optimizer with a bad mesh which would result in not accurate results. The best mesh has been designed for an angle of 15 degrees in a centered position (not too close nor far). Hence, the mesh generated will not be perfect for all the cases but will be acceptable for the majority of them.

On the Figure 3.12 it is shown the adapting mesh to different positions, $\delta = 5^o$ and $\delta = 30^o$.



(A) (B)

FIGURE 3.12: Different automated domains around changing flap positions.

However, there are still positions that can not be achieved due the fail of the mesh. It has been assured that these positions are not the most optimum ones to not affect the optimization process. Nevertheless, because of this, restrictions will be needed when optimizing in order to avoid problematic meshes.

In the Figure 3.13 it is shown one of the limits that would produce bad results. There are also flap positions that Salome does not allow to open as the geometry requested is not possible to create. Those limits have also been repaired or, in the case that no reparation was possible, removed from the testing range.

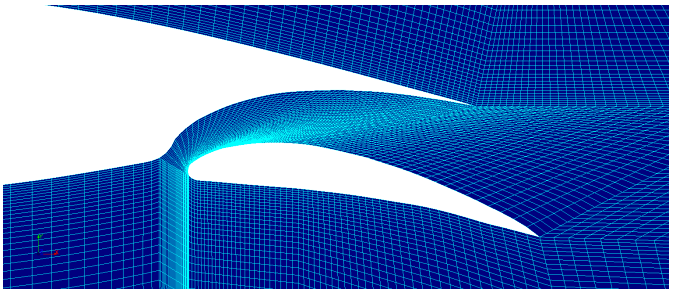
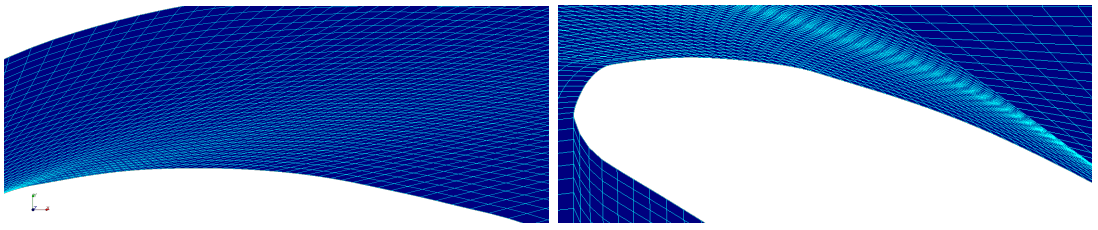


FIGURE 3.13: Flap limit due to mesh failure.

The process to obtain a mesh valid for all the desired cases has been long. Many errors have been encountered with the surfaces not adapting the changes and the cells having orthogonality problems increasing their skewness. These problems are important to be detected for the process to function correctly. For instance, highly skewed cells can lead to stability problems for CFD solvers. In the Figures 3.14a and 3.14b failure meshes are closely shown to observe their highly skewed cells.



(A) (B)

FIGURE 3.14: Highly skewed cells in limit flap positions.

After the discretisation is finished, the boundaries are prepared for their later extrusion and assignation. All the edges that will be inlet, outlet or wall must be defined. When these edges from 2D mesh become extruded, its corresponding 2D cells will be defined for the boundary conditions.

Before extruding the mesh from 2D to 3D all the meshes from each region must be united into one compound mesh, with its chosen tolerance. With the compound it is now possible to extrude the mesh and create the hexahedras. The compound can be also done after the extrusion but this would increase the time of the process.

The extrusion length is of no importance as there will be no flow changes in that direction. However, if the size is too high, OpenFOAM will show problems with the aspect ratio of the cells.

Afterwards, the boundary conditions can be created as will be explained in the section 3.4, and finally the mesh is automatically exported in *.unv* type file inside the OpenFOAM folder. This mesh will be later transformed into the OpenFOAM format.

The Salome part is finished and the program does not need to intercede any longer on the same individual, so it has been used a call to close the program in order to automate the process.

3.3.3 Mesh from Salome to OpenFOAM

Once the Salome program is closed, the command `ideasUnvToFoam` turns the *.unv* mesh to a *polyMesh* inside the constant folder, this mesh is the one OpenFOAM understands, and all its components are the ones shown in Figure 3.15. [12]

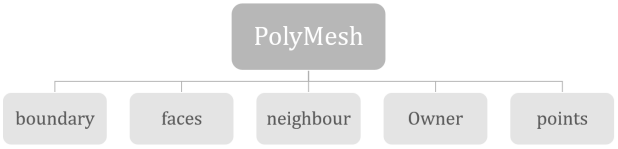


FIGURE 3.15: PolyMesh folder.

Faces

File where all the mesh faces are defined as the number of points composing it, for instance, 4(751 288 21479 22013) where 4 is the number of the cell face and the rest are the four points. The ordering of point labels in a face is such that each two neighbouring points are connected by an edge.

Neighbour

Each face is therefore assigned an 'owner' cell and 'neighbour' cells so that the connectivity across cells is defined. In this file is placed the list of neighbour cell labels.

Owner
File with the list of owner cell labels, following the description of the Neighbour file.

Points
The points are compiled into a list as (0.002375 0.0040356 0) describing the cell vertices where the first vector in the list represents vertex 0, the second vector represents vertex 1, etc. The points are 3D located with a vector in units of metres (m).

Boundary
A list of patches, containing a dictionary entry for each patch, declared using the patch name (wall, empty...). The meaning of different patches and how this file needs to be modified is explained after in the section 3.4.

3.3.4 Mesh quality

Once the mesh is fully created and exported, the command `checkMesh` is run in order to see failures in the mesh like high aspect ratio, skewness, non-orthogonality, etc.

As it is shown on the Listing 3.2, if there is a failure in the mesh this tool detects it and saves the name of the cells with the error in a file. The error is marked with * like shown on the extracted code below. The higher the mesh error the more *** appear. As can be also seen, if the parameters are acceptable, an OK appears on the side.

```
Min face area = 6.5189e-09. Max face area = 0.00100142.  Face area OK.\\

Min volume = 3.25945e-11. Max volume = 5.00709e-06.  Total volume = 0.00732491. ↩
     Cell volumes OK.\\

Mesh non-orthogonality Max: 84.1934 average: 25.8112\\

*Number of severely non-orthogonal (> 70 degrees) faces: 1320.\\

    <<Writing 1320 non-orthogonal faces to set nonOrthoFaces
```

LISTING 3.2: CheckMesh utility by OpenFOAM

A non-orthogonality above 70 degrees needs treatment with the `nonOrthoCorrectors` utility in the OpenFOAM case file *fvSolution* or numerical schemes in the *fvSchemes* file. If it is above 90 degrees the mesh can not be used.

Errors can also be identified in Salome, where the high skewness cells are highlighted. There is also the possibility to see high aspect ratio cells, control the area or angle of the cells and their taper ratio.

The quality measurements of the mesh are of high importance as they influence directly the behaviour of the computation. A high aspect ratio on the cells can decrease the convergence speed stability, higher skewness values can reduce the accuracy of the results, and non-orthogonality can lead to stability problems.

This tools have been used when choosing the maximum and minimum values of dx, dy and δ, to know where the mesh is not considered valid anymore and which aspects have to be changed.

3.3.5 Mesh limitations

Limits of the flap movements have to be determined, as the optimizer will choose the values of dx, dy and δ between the limits. When searching for this border positions it has been noticed that these are not independent for each variable (dx, dy or δ), the limits of dy are directly conditioned by the deflection δ.

The limits have been found testing the mesh quality for dy maximums and minimums for a range of deflections δ. The dx has been limited between two fixed values for all dy and δ.

When the optimizer RMOP2.0 is set up (Section 1.2) a constraints file can be created to restrict these conditioned limits. However, the optimization process would be slower as it would have to check each time new positions are proposed, if they fulfill the restrictions or not.

For this reason, no use of constraints has been made and the dy variable has been limited in a range values between [-1,+1]. Later this values will be interpolated with the real dy limitation values for each range of deflection δ. The script implementing the restrictions and interpolations can be encountered in the Section 6.2.3.2 and in the Appendix A.1.

3.4 Boundary conditions

In a CFD analysis, it is needed to assign boundary conditions to define how the case operates. Inappropriate boundary conditions would lead to physically incorrect predictions, and in many cases solver failure. These boundary conditions are created by assigning each boundary to a number of cells and defining its geometrical constraint (wall, inlet, outlet, etc). Afterwards, the boundary conditions are specified in the field files inside time directories. These fields are velocity, pressure and turbulence parameters and will be assigned on the initial conditions of the case 3.4.2.

3.4.1 Boundaries

The boundaries or patches in the case of study will be wall, for the solids, empty, for the faces on the front and back, and patch for inlets and outlets.

Inlet and outlet

Inlets and outlets are assigned with the patch type, which contains no geometric or topological information about the mesh.

Walls

Patch assigned to the solid walls, airfoil and flap in this case, it is required for turbulence modelling.

Front and back

The empty patch is used for solving in 2 dimensions a 3D geometry like the one in this case, an extruded 2D mesh. It is needed to specify the empty condition on each patch whose plane is normal to the 3rd dimension, front and back patches for this case, as no solution is required in the 3rd dimension.

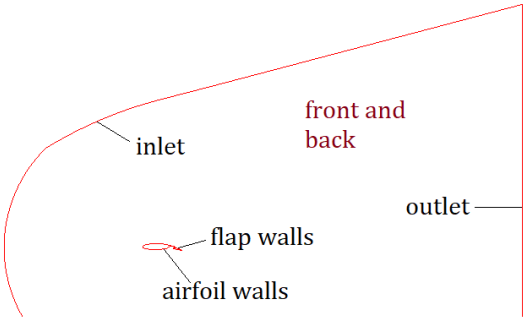


FIGURE 3.16: Patches of the case.

3.4.1.1 Creation of the boundaries

The first thing to do in order to define the boundaries is to make groups of the cells conforming each boundary. This can be made with Salome or after, with the *boundary* file in *polyMesh*.

In the automated process it is done with Salome tools except for the front and back boundaries because had so much number of cells that neither the interface nor the script could process them. This does not happen in the boundary file as not all the faces are written down, the group is only defined by the first one and the number of faces, this is thanks to the faces file in *polyMesh* that numerates the faces in order.

When making the boundaries with Salome, groups of cells need to be made from the extrusion, and the old groups created automatically need to be deleted if not the cells (or faces) will be defined twice.

The walls, displayed in Figure 3.17, will be separated in `airfoil_intrados`, `airfoil_extrados`, `fowler_intrados` and `fowler_extrados` for the later plot of the pressure coefficient. Also the groups `inlet` and `outlet` are created. The `front` and back are left in the groups created automatically from the extrusion, one for each surface mesh.

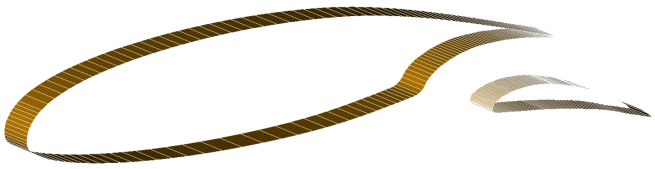


FIGURE 3.17: Airfoil and Fowler flap patches.

When the polyMesh is created, the boundaries made with Salome are correctly made. The ones for the front and back are easy to modify, putting together all the boundaries that form the front and back boundaries.

The *boundary* file has also to be modified for a reason, when the boundary patches are created are not correctly assigned, one must change them. As explained in 3.4.1, the airfoil and Fowler flap patches are needed to be assigned as *walls*, the front and back ones as *empty* and the inlet and outlet as *patches*.

A segment of the resulting *boundary* file for this case showing one type of each patch can be seen in the Listing A.2.2.1, the complete file can be found in the Appendix A.2.2.1.

```
8
(
     front
    {
        type            empty;
        nFaces          31701;
        startFace       94673;
    }
    airfoil_extrados
    {
        type            wall;
        nFaces          83;
        startFace       126374;
    }
    inlet
    {
        type            patch;
        nFaces          383;
        startFace       126682;
    }
)
```

LISTING 3.3: Boundary file

The number on the top is the amount of patches, 8 for this case. The `nfaces` is the number of faces of the patch and `startFace` the name of the starting face.

3.4.1.2 Automation of the boundaries

For the optimization, even though the flap position is changing the number and name of the cells for all changes are the same. The boundary file can be always the same while the points file in polyMesh changes. This way, the patches are assigned correctly in each different position.

3.4.2 Initial conditions

The boundary conditions have to be defined in the initial conditions files (p, U and turbulence properties) for each patch defined in the *boundary* file, see section 3.4.1. These initial conditions files are placed in the time directory, usually called *0* folder for starting at time=0.

The pressure and velocity files are going to be explained in this section while the turbulence files will be explained in the turbulence model section 4.4 as the file description depends directly on the turbulence model chosen.

3.4.2.1 Velocity (U)

The first thing needed to be defined is the class of the *FoamFile*. As velocity is a vector field its class needs to be defined as `volVectorField`.

Afterwards, the dimensions are represented as a vector where each component represents a unit: `[Mass(kg) Length(m) Time(s) Temperature(K) Quantity(kgmol) Current(A) Luminous intensity(cd)]`. The values are the unit exponent, therefore, velocity must be represented as $m^1 s^{-1}$. After the dimensions, the internal field value is described, it can be `uniform` or `nonuniform`. When uniform is chosen, the values of this velocity field are specified in each direction.

The first part described of the velocity file is shown in the Listing 3.4. In the case of study, the initial velocity is $U = 5m/s$ and the angle of attack $\alpha = 4^o$. As the airfoil has been placed geometrically with an angle of 0^o, this angle has to be now set changing the direction of the velocity. This way, the resultant values of U_x and U_y are the ones displayed.

```
FoamFile
{
    version 2.0;
    format binary;
    class volVectorField;
    location "";
    object U;
}
dimensions [0 1 -1 0 0 0 0];
internalField uniform (4.987820251 0.3487823687 0.);
```

Listing 3.4: Velocity field file (U)

The second part of the file is where the patches are assigned a boundary condition of velocity. The `boundaryField` is a dictionary containing a set of entries whose names correspond to the boundary patches listed before in the *boundary* file.

The *type* entry describes the patch field condition for the field. For the front and back patches the type is *empty* as no information is needed in the 3rd direction (Z). On the walls (airfoil and flap), as the velocity will always be zero in the surface, the type will be a *fixed value* of uniform 0 velocity, it also can be applied the `noSlip` condition. Finally, on the inlet and outlet patches the velocity will be also a *fixed value* that will not change during the simulation, with the entering velocity of 5m/s and $\alpha = 4^{o}$.

A small part of the velocity file is represented as follows for the same patches that have been defined in the boundaries section 3.4.1.

```
boundaryField
{
    front
   {
      type empty;
   }
    airfoil_extrados
   {
      type fixedValue;
      value  uniform (0. 0. 0.);
   }
    inlet
      {
      type fixedValue;
      value  uniform (4.987820251 0.3487823687 0.);
   }
}
```

LISTING 3.5: Velocity field file, part 2 (U)

3.4.2.2 Pressure (p)

The pressure file is similar to the velocity one in terms of format. The pressure field is a scalar field so its *Foamfile class* will be `volScalarField`. The pressure defined in the file is, in reality, the kinematic pressure p/ρ and its dimensions are $Pa/kg/m^3$ or, what it is the same, $m^2 s^{-2}$. As the case is treated with relative pressure the internal field uniform value will be of 0 m^2/s^2.

Regarding the boundary conditions on the patches, the front and back boundaries will have again an *empty* type, and the patches inlet, outlet and walls a zero gradient of pressure.

```
dimensions [0 2 -2 0 0 0 0];
internalField uniform 0.;

boundaryField
{
    front
   {
         type empty;
   }
    airfoil_extrados
    {
      type zeroGradient;
   }
      inlet
   {
      type zeroGradient;
   }
}
```

LISTING 3.6: Pressure field file (p)

3.4.2.3 Automation of initial conditions

As the initial conditions are defined with the patches of the *boundary* file, as long as this file is correctly made in every flap change, the initial conditions will be appropriate. These files do not change, are the same for all the iterations.

3.5 Summary

The pre-process needed before solving the case has been thoroughly explained in this Chapter.

The first step has been the creation of the geometry of the airfoil and flap in Salome9.2.1, and after, its automation for the flap changes. The following step has been the design and creation of the outer domain and sub-domains, in order to facilitate the meshing process. These have been modified several times until its shape has been able to adapt to the flap changes.

To create a correct mesh, the meshing parameters have been carefully specified for the different regions of the domain. These are the dimensionless wall distance, its related first cell height, the cell width and the growth rate. Afterwards, the mesh has been implemented in Salome and it has been automated. It has to be noted that the size of the cells has been chosen sufficiently small to give appropriate results, but as high as possible inside the acceptable limits to avoid a high computational time not bearable for an optimization during the Thesis period of time.

Regarding the automation of the mesh, the changing positions of the flap have caused mesh failures that have been detected thanks to the mesh checks provided by Salome and OpenFOAM. The mesh has been modified to eliminate all the possible conflicting cells like highly skewed cells that can lead to stability problems or undesirable aspect ratios that can fail to simulate the flow movement.

The mesh failure positions that have not been able to correct without affecting negatively other positions have been responsible of the limitation flap positions. It has been assured that these limits are not prejudicial to the optimization.

Finally, the exportation process of the resulting mesh and its conversion to OpenFOAM has been explained with its automation process. And the mesh boundaries have been defined.

Once the meshing process is finished, to have all the process set up it is needed to establish the boundary conditions, composed by the specification of the patches and its initial conditions together with the internal field ones, for velocity (U), pressure (p) and the turbulence coefficients.

Chapter 4

Solving the case

Having completed the pre-process part, it is proceeded to solve the fluid around the geometry. OpenFOAM enables the user to decide on mostly all the parameters of the solving process, this is why the preparation of the solving has to be carefully performed.

First of all, the OpenFOAM solver that will be used has to be selected, based on the physical properties of the case explained on the Chapter 2. When deciding the solver, some clarifications have to be made about its solving algorithms in order to understand its functioning. The solver will be chosen both for its reliability on the optimization process and its minimum computational time.

A part from the solver, the turbulence model has also to be defined, which will depend on the desired results of the CFD and the computational cost.

4.1 OpenFOAM solvers

As mentioned in the *Case description*, Chapter 2, being a subsonic case with a low number of Mach, the flow will be solved as incompressible. A list of incompressible solvers has been made following the *OpenFOAM Guide* [13] in order to choose the most suitable one for the case.

4.1.1 Incompressible flow solvers

icoFoam
Transient solver for incompressible, laminar flow of Newtonian fluids.

nonNewtonianIcoFoam
Transient solver for incompressible, laminar flow of non-Newtonian fluids.

simpleFoam
Steady-state solver for incompressible flows with turbulence modelling, using the SIMPLE algorithm, the algorithm is explained in the Section 4.2.

pisoFoam
Transient solver for incompressible, turbulent flow, using the PISO algorithm, explained in the Section 4.2.

pimpleFoam
Transient solver for incompressible, turbulent flow of Newtonian fluids on a moving mesh, using the PIMPLE (merged PISO-SIMPLE) algorithm.

porousSimpleFoam
Steady-state solver for incompressible, turbulent flow with implicit or explicit porosity treatment and support for multiple reference frames (MRF), using the SIMPLE algorithm.

SRFSimpleFoam
Steady-state solver for incompressible, turbulent flow of non-Newtonian fluids in a single rotating frame, using the SIMPLE algorithm.

SRFPimpleFoam
Large time-step transient solver for incompressible, turbulent flow in a single rotating frame. Uses the PIMPLE algorithm.

As the flow is turbulent, all the laminar solvers can be discarded unless the case is solved with direct numerical simulation (DNS), which is not the case. Besides, as air is a Newtonian fluid, all non-Newtonian fluid solvers can also be dismissed together with the solvers for particular cases such as porosity.

Considering these statements, the resultant solvers are the ones solving steady (SIMPLE) or unsteady (PISO and PIMPLE) incompressible turbulent flows, just as the flow in the case of study. These solution algorithms are going to be discussed and compared to end up with the correct solver.

4.2 Solution algorithms

Having the discretised momentum equations, if the pressure is known, the equations can be solved to obtain the velocity fields. As the pressure field is unkwnown, it is needed an algorithm in order to obtain the solution fields. The velocity field obtained through the pressure will satisfy continuity if the calculated pressure is correct, this is why the equations that will be used for the resolution method will be the discretised form of the momentum (1.2) and continuity (1.1) equations.

4.2.1 SIMPLE (Semi-Implicit Method for Pressure Linked Equations) algorithm

The SIMPLE algorithm, put forward by Patankar and Spalding (1972), is an iterative algorithm proven to be efficient in solving steady state problems using a predictor-corrector loop and under-relaxation factors. It uses the pressure-based method.

At the beginning of the SIMPLE process, the pressure field is initially guessed p^*, and with it, the discretised momentum equations for x- and y-directions are solved, obtaining u^* and v^*.

Now the corrections of pressure p' and velocities u' and v' are defined as the difference between the real values and the ones guessed.

$$p = p^* + p', \qquad u = u^* + u', \qquad v = v^* + v' \tag{4.1}$$

If the correct pressure field p is entered to the discretised momentum equation, with some approximations, the correct velocity field equations are obtained (u, v) depending on p. The procedure can be found in (H. K. Versteeg and W. Malalasekera, 2007) [2].

The continuity equation has to be also taken into consideration. Substituting the discretised equations for the velocities (u,v) into the discretised continuity equation, the pressure correction p' equation results. The pressure correction is susceptible to divergence, hence, under-relaxation factors α are used during the iterative process, and new pressures are obtained

$$p^{new} = p^* + \alpha_p p' \tag{4.2}$$

The velocities are also under-relaxed, and therefore under-relaxation factors are also applied

$$u^{new} = u^* + \alpha_u u', \qquad v^{new} = v^* + \alpha_v v' \tag{4.3}$$

The usage of these relaxation factors will be later explained when preparing the OpenFOAM case, in the Section 5.3.3.

Once pressure and velocities are corrected, the other fields such as temperature and turbulent kinetic energy can be calculated.

If the pressure and velocity fields do not satisfy the specified residual tolerance, which is the difference between the guessed or previous iteration values and the new ones calculated, the algorithm proceeds to the next iteration, substituting

$$p^* = p, \qquad u^* = u, \qquad v^* = v \tag{4.4}$$

The SIMPLE algorithm explained is represented in the flux diagram in the Figure 4.1,

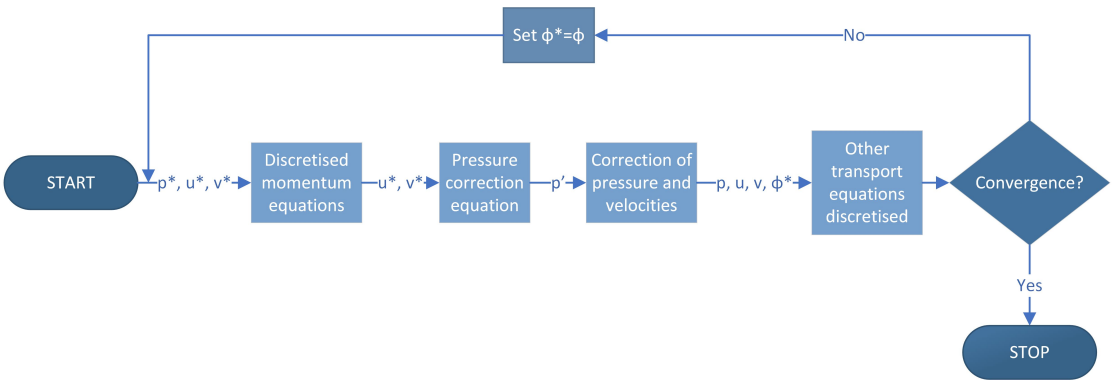


FIGURE 4.1: Pressure and velocity fields solved using the SIMPLE algorithm.

4.2.2 PISO algorithm (Pressure-Implicit with Splitting of Operators)

The PISO algorithm (Pressure-Implicit with Splitting of Operators) is an extension of the SIMPLE algorithm to solve the Navier-Stokes equations without iteration in steady flows. The algorithm is going to be explained for steady flows.

PISO has the same procedure that SIMPLE but with a second corrector step. The twice-corrected pressure field is obtained from

$$p^{***} = p^{**} + p'' = p^{*} + p' + p'' \tag{4.5}$$

In the non-iterative calculation of steady flows the pressure field p^{***} and the velocity field u^{***} and v^{***} are considered correct. For an iterative steady state calculation the iterative process follows the algorithm diagram on Figure 4.1.

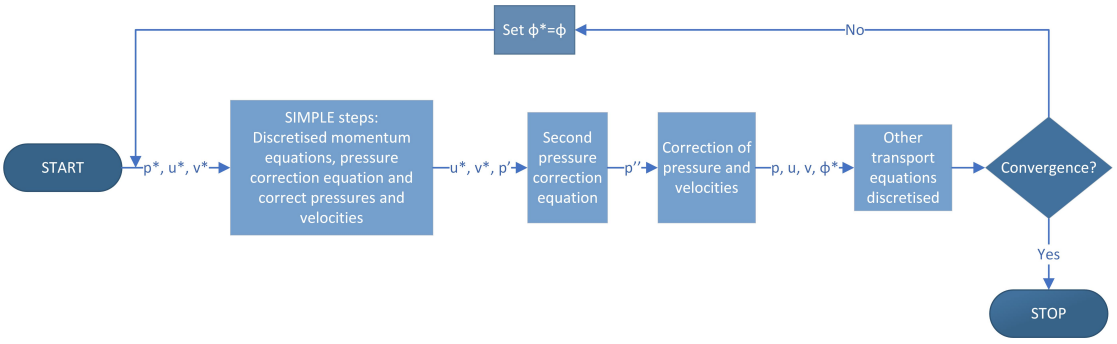


FIGURE 4.2: Pressure and velocity fields solved using the PISO algorithm.

The pisoFoam solver adds to this algorithm the variation in time, it is therefore essential to carefully discretise the time in order to capture the fluid flow correctly, as it is explained on the Section 5.3.1.1.

4.2.3 PIMPLE algorithm

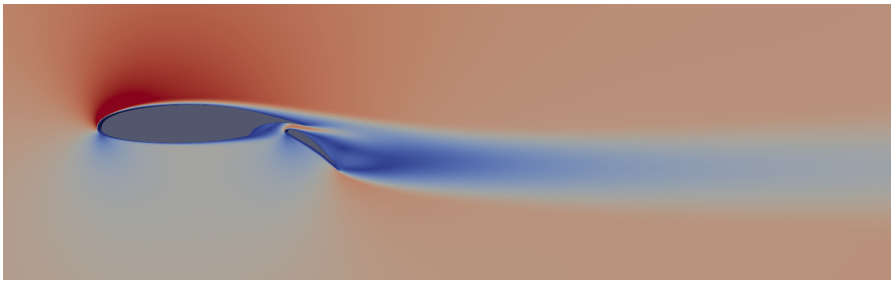
Is a combination of PISO and SIMPLE. Better stability is obtained from PIMPLE over PISO being able to work with large time steps or when the nature of the solution is inherently unstable.

4.3 Study of pisoFoam vs. simpleFoam for a comparative analysis

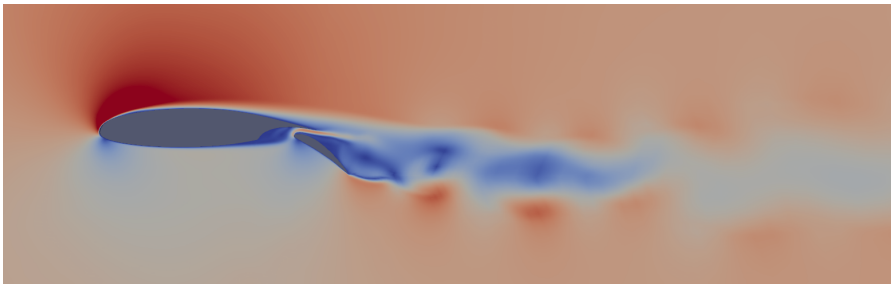
One of the main optimization requirements is to evaluate a large number of possibilities in order to produce better results, this matter has resulted in a problem when coupling optimization with CFD as the computation of the solution takes a long time. For this reason, a study has been carried out between the calculations made with pisoFoam and simpleFoam in order to see if simpleFoam could determinate the same best position as pisoFoam needing less computational time. All this knowing that the value of the results will not be as accurate as the ones with pisoFoam, which takes transient effects into account.

The study has been based on the calculation of different positions with both solvers. As the deflection angle is the parameter that affects transient effects the most, the test will start with the calculation of different angles. The solutions obtained by both solvers are not expected to be the same, however, the relations of lift and drag coefficients between the different angles are required to be similar.

The fist angles that have been tested are 30, 22, 15, 7 and 0 degrees, for a fixed x and y position. The difference between the solvers is noted considerably in the vortex shedding as can be seen for the 30 degree deflection simulation represented on the Figure 4.3.



(A)



(B)

FIGURE 4.3: Simulation results for δ=30º with (a) simpleFoam and (b) pisoFoam.

The coefficients are directly obtained from the *postProcessing* folder explained in the Chapter 5, how these coefficients are extracted is detailed in the coupling Section 6.2.3.5.

The resultant values of the force coefficients at the flap deflections mentioned are shown on the Table 4.1.

		30º	22º	15º	7º	0º
simpleFoam	Cl	2.017177	1.942980	1.690103	1.559536	1.258370
	Cd	0.153499	0.130904	0.098141	0.081545	0.067721
pisoFoam	Cl	2.041178	2.007720	1.716830	1.609086	1.293621
	Cd	0.146296	0.115598	0.075921	0.067151	0.056938

TABLE 4.1: Comparison of aerodynamic coefficents for different flap deflections with simpleFoam and pisoFoam.

As can be seen, the influence of changing the deflection angle has been correctly interpreted by simpleFoam in terms of comparison with other angles. As expected, the values do not coincide, so in case simpleFoam is used the values that will be obtained for the optimized position should be simulated again at the end of the optimization in order to obtain more accurate results.

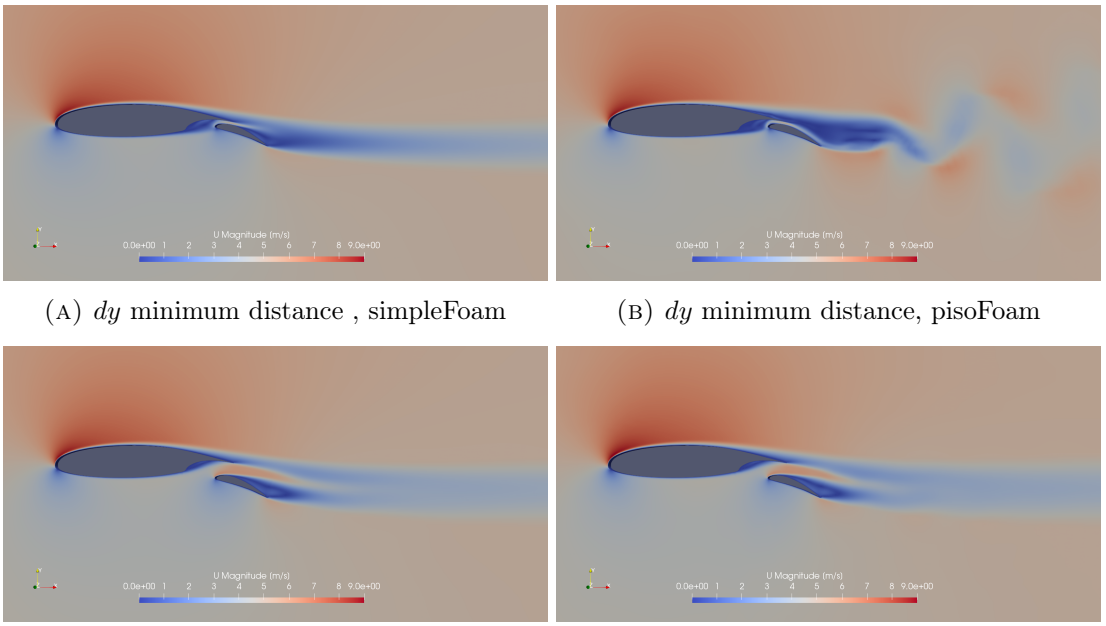
With the aim of reassuring the simpleFoam capacity of comparison, new positions have been tested for a fixed angle of 15 degrees with the maximum positions of dx and dy along with the middle one. The minimum dy position is a distance of 2%c from the airfoil and the maximum a 8%c distance. The dx minimum is placed at 5%c from the airfoil and the maximum at 15%c. These limits shown on the Table 4.2 have been a consequence of the mesh, in order to have a good quality in all positions.

		dy			**dx**		
		min	center	max	min	center	max
simpleFoam	Cl	1.71875	1.69010	1.73566	1.69201	1.69010	1.80319
	Cd	0.12346	0.09814	0.10720	0.10299	0.09814	0.10086
pisoFoam	Cl	1.66495	1.71683	1.79409	1.76167	1.71683	1.84703
	Cd	0.14260	0.07592	0.09815	0.08834	0.07592	0.08657

TABLE 4.2: Comparison of aerodynamic coefficients for different flap positions with simpleFoam and pisoFoam.

The same differences as before are encountered in this test, lower values of Cl and higher values of Cd in simpleFoam compared to pisoFoam. These may be due to the absence of the vortexes on the downstream flow. Nevertheless, if the values of the different positions are compared between them only considering which are better, the outcome of simpleFoam and pisoFoam is the same.

The simulation results for the *dy* case are displayed in the Figure 4.4. It has to be noted that the use of simpleFoam instead of pisoFoam has higher effects when the flap is close to the airfoil, as the downstream flow has a more turbulent behaviour. However, when the flap is far from the airfoil, the velocity field is similar.



(A) *dy* minimum distance , simpleFoam

(B) *dy* minimum distance, pisoFoam

(C) *dy* maximum distance , simpleFoam

(D) *dy* maximum distance, pisoFoam

FIGURE 4.4: Simulation results for *dy* minimum and maximum distances, solved with simpleFoam and pisoFoam.

Summarizing, not having into account the transient effects and using the SIMPLE algorithm increases the error of the solution values but still has reliable results for comparison. For this reason and considering the decrease of computational time that implies implementing simpleFoam, it has been chosen as the solver to use in the process. Otherwise, the optimization process would be impossible to run with a large number of iterations in the time available for the Thesis.

Nevertheless, it is thought to improve the optimization in future work adding a pisoFoam solution. Moreover, the solution of the actual optimization made with simpleFoam could be used as the initial guess for further optimizations.

4.4 Turbulence models

The computational time does not permit solving the turbulence by means of the governing equations direct solution. Therefore, approximate methods for solving turbulence have been used.

First, a brief introduction about the turbulence resolution is needed. The methods can be grouped in three main categories:

- **Direct numerical simulation (DNS)**: Navier-Stokes are solved on grids sufficiently fine to resolve the Kolmogorov length scales at which dissipation of energy takes place, and with time steps sufficiently small to resolve the fastest fluctuations. The computational cost is extremely high.
- **Large eddy simulation (LES)**: intermediate form of turbulence calculation which catches the behaviour of the larger eddies and rejects the smaller ones. Unsteady flow equations must be solved. The computational cost is large.
- **Reynolds-averaged Navier-Stokes equations (RANS)**: the resolution is focused on the effects of turbulence on mean flow properties. Extra terms appear in the averaged flow equations due to turbulence fluctuations, these terms are modelled with turbulence models: $k-\epsilon$ model, $k-\omega$ model, Spalart Allmaras, Reynolds stress model, etc.

As the solution of fluid flow in the case is only needed for the optimization, paying exclusively attention to compare the aerodynamic coefficients, the accuracy provided by DNS and LES methods will be renounced due to its high demanding computational cost. This is why a RANS turbulence model has been selected, as it computes sufficiently right the aerodynamic forces while it will be able to solve the different cases without lasting extremely long.

Regarding the RANS turbulence model to use, it has been made a research to know the capacities and limitations of the avaliable OpenFOAM models. The Spalart-Allmaras model has been selected due to its specially good behaviour on external flows around aerodynamic geometries while its running computational time is lower than some of the other models, as it uses only one extra transport equation. The transport equation that Spalart-Allmaras model involves is for the kinematic eddy viscosity ν, parameter that will have to be specified in the turbulence initial conditions on the OpenFOAM case, Section 5.1.1.

4.5 Summary

The different OpenFOAM solvers have been discussed together with the algorithms for solving the governing equations (SIMPLE and PISO).

Afterwards, a comparison has been made between the most suitable solvers (simpleFoam and pisoFoam), and simpleFoam has been chosen since it has shown reliable results for the optimization. Nevertheless, its limitations on the results are acknowledged and the decision is made only for the first optimization tests. In a future work, after finding the best results of the optimization process, the pisoFoam will be used to verify the best results or optimize them again, with a higher accuracy.

Moreover, the turbulence models have been briefly defined and the Spalart-Allmaras RANS model has been selected due to its proper behaviour on aerodynamic geometries while it has a low computational cost. Also, more accurate turbulence models could be used in further improvements of the optimization, as long as the mesh was refined.

Chapter 5

OpenFOAM case setup

Chosen the solver and turbulence model to use for the CFD simulation, the OpenFOAM case folders can be prepared and its containing files specified with the solver parameters. These will be explained in this section and will be focused on the `simpleFoam` solver. Once the solving is set up, its automation process will be commented.

5.1 Initial conditions

The initial conditions from which the case will start to compute the equations are placed in the time directory, it is usually called 0 for a computational time of $t = 0$ but a simulation can start also from a time directory with the solution of a more advanced computed result, like for instance t=50. Here, the boundary conditions for each patch are defined (see section 3.4 for patch description).

In this directory, which will be 0 for the optimization case, are located the files of velocity (U), pressure (p) and turbulence parameters, which are *nut* and *nuTilda* for *Spalart Allmaras* as the turbulence model. The U and p files have already been explained in the boundary conditions section, 3.4.2.1 and 3.4.2.2 respectively, the *Spalart Allmaras* parameters will be explained below.

5.1.1 Turbulent kinematic viscosity ν_t (nut)

The turbulent kinematic viscosity is a scalar field with its dimensions being m^2s^{-1}. The boundary conditions for the ν_t have been extracted from the *NASA Turbulence Modeling Resource* [14].

The ν_t value in the wall must be 0, while the ν_t in the far field has a value referenced to the ν_∞ ($\nu_\infty = 1.81 \times 10^{-5}$), as the result is small the value has been defined as 0.

```
dimensions      [0 2 -1 0 0 0 0];
internalField   uniform 0;
boundaryField
{   front
      {
        type            empty;
      }
    airfoil_extrados
      {
        type            nutLowReWallFunction;
        value           uniform 0;
      }
    inlet
      {
        type            calculated;
        value           uniform 0;
      }
}
```

Listing 5.1: Turbulent kinematic viscosity file (ν_t).

In the Listing 5.1 the ν_t values for each patch type are specified. The `font` and `back` patches will have a `type empty` nut while the other patches will have a `uniform value` of 0. The nutLowReWallFunction sets nut to zero, and provides an access function to calculate y^+.

5.1.2 Spalart-Allmaras model modified viscosity ν' (nuTilda)

The Spalart-Allmaras model modified viscosity is also a scalar field with the same dimensions as ν_t or ν_∞ $[m^2 s^{-1}]$, and the information about its boundary conditions has been also extracted from the *NASA Turbulence Modeling Resource*.

As it happens with ν_t, the value of ν' on the walls has to be 0. The ν' on the far field it is defined to be between $3\nu_\infty$ and $5\nu_\infty$ ($\nu_\infty = 1.81 \times 10^{-5}$), so the value has been considered 5×10^{-5}.

The `front` and `back` patches are, once again, defined as empty. The file is shown in the Listing 5.2.

```
front
  {
    type            empty;
  }
airfoil_extrados
  {
    type            fixedValue;
    value           uniform 0;
  }
inlet
  {
    type            freestream;
    freestreamValue uniform 0.000005;
  }
```

Listing 5.2: Spalart-Allmaras modified viscosity (ν').

5.2 Constant properties

Inside the *constant* directory are the files in which the thermophysical and turbulence settings are specified as well as the mesh properties.

5.2.1 polyMesh

The files composing the *polyMesh* directory are the ones that contain all the information about the mesh and its patches. See Section 3.3.3 for more information about it.

5.2.2 Transport properties

In the solvers that do not include heat, a model for the kinematic viscosity ν has to be specified. There are different models that can be chosen, explained in the *OpenFOAM User Guide* [13]. The one selected for this case is the Newtonian model which assumes ν is constant.

Also the values of density (`rho`) and kinematic viscosity (`nu`) are established, which, for the case are:

$$\rho = 1.225 \qquad \nu = 1.81 \times 10^{-5} \tag{5.1}$$

The *transportProperties* file is presented in the Listing 5.3

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}

transportModel  Newtonian;

rho             [1 -3 0 0 0 0 0] 1.225;

nu              [0 2 -1 0 0 0 0] 0.0000181;
```

Listing 5.3: Transport properties file.

5.2.3 Turbulence properties

If the solver chosen includes turbulence modelling, the *turbulenceProperties* dictionary is read. The first thing needed to be specified is the `simulationType` which can be:

- **Laminar**, uses no turbulence models. To use the direct numerical solution (DNS) the type should be defined as laminar, since the equations are solved directly with no modelling of turbulence.
- **RAS**, uses Reynolds-averaged simulation modelling.
- **LES**, uses large-eddy simulation modelling.

As explained in the turbulence section 4.4, the RAS model has been chosen for the case, with the Spalart Allmaras model type. The RAS model requires the following entries:

- `RASModel`, name of RAS turbulence model which is Spalart Allmaras for the case under study.
- `turbulence`, switch to turn the solving of turbulence modelling.
- `printCoeffs`, switch to print model coeffs to terminal at simulation start up.

The file *turbulenceProperties* is the one in the Listing 5.4.

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      turbulenceProperties;
}

simulationType RAS;

RAS
{
    RASModel        SpalartAllmaras;

    turbulence      on;

    printCoeffs     on;
}
```

LISTING 5.4: Turbulence properties file.

5.3 System

The *system* directory contains the files in which the discretization schemes, parallelisation settings, solver settings and control settings are specified.

5.3.1 ControlDict

The *controlDict* dictionary controls the solver's behaviour, thus, the first thing needed to be specified is the OpenFOAM solver in `application`, which in this case will be *simpleFoam* as justified in the section 4.2.

```
application     simpleFoam;
```

The mandatory entries for the file are the time control and the writing settings. Extra functions can be added at the end in order to obtain extra results.

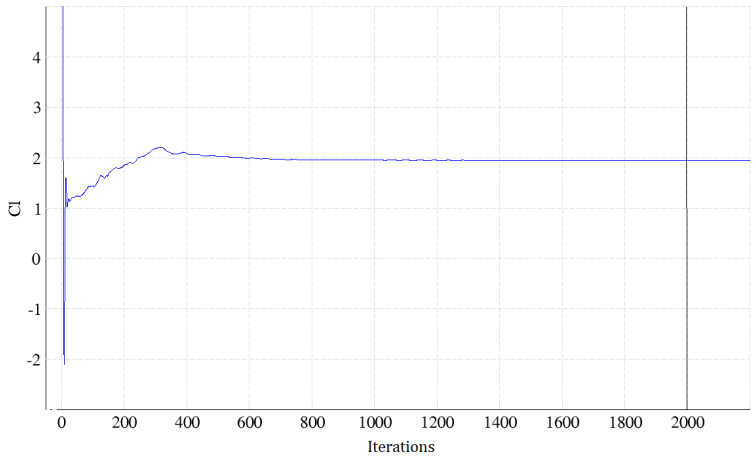
5.3.1.1 Time control

The time settings that have to be configured are the first time from which the simulation will start (`startFrom`), the computational time at which the simulation will stop (`stopAt`), which for simpleFoam is the number of iterations, and the increment of time from the start time until the stop time (`deltaT`), also known as time step.

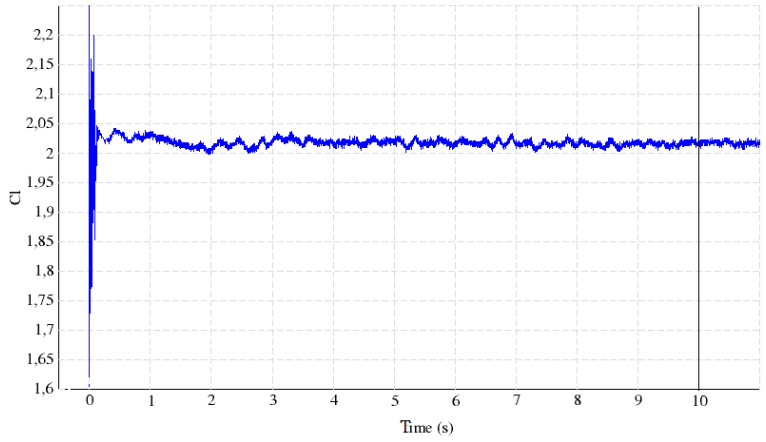
The start time has been defined as *latestTime* in case the simulation is wanted to start from a higher value than $t = 0$. The starting time will be set by the number of the time directory placed in the case folder.

The last time has been defined as *endTime* which has to be specified as a number in `endTime` keyword entry. The end time has been chosen after several tests looking at the convergence of the solution. This convergence time is different for solving with different solvers (*pisoFoam* and *simpleFoam*).

As can be seen on the Figure 5.1, for the simpleFoam case the solution is converged at 2000 "seconds", as said before in simpleFoam the time represents the number of iterations as variations in time are not computed. The pisoFoam solution is converged at 10 computational seconds. This does not mean that pisoFoam does converge earlier, its time step is much lower as will be explained next.



(A)



(B)

FIGURE 5.1: Lift coefficient convergence for δ=22º, dx=0 and dy=0, with (a) simpleFoam and (b) pisoFoam.

The time step is an important parameter in numerical analysis of time integration schemes, such as the pisoFoam solver. The time step must be less than the time for the wave to travel from one grid point to its adjacent one, in order to capture the fluid flow. This necessary condition for convergence is limited by the *Courant–Friedrichs–Lewy (CFL) number*, a dimensionless number which considers the time step, the cell dimensions and the velocity of the fluid:

$$C = \frac{u_x \Delta t}{\Delta x} + \frac{u_y \Delta t}{\Delta y} \leq 1 \tag{5.2}$$

being u the velocity in each direction, Δt the time step and Δy and Δx the cell sizes.

Considering the Δy and Δx of the mesh and using the free-stream velocity as an approximation it results,

$$\frac{4.9878\Delta t}{5.1 \times 10^{-4}} + \frac{0.3488\Delta t}{5.1 \times 10^{-4}} \leq 1 \tag{5.3}$$

So an approximated maximum time step for the pisoFoam case is needed to be under $\Delta t = 9.56 \times 10^{-5}$. A more accurate result will be obtained during the simulation, as the CFL number appears in each iteration and can be seen if increases until 1 or more, then, the time step can be reduced.

The simpleFoam solver has no time integration schemes as it is a steady-state solver. For this reason, the CFL number does not have to be considered and the value of `deltaT` can be 1. This `deltaT` and the `endTime` still have meaning for the solver, they represent the number of iterations, and must be such for the solution to converge.

The time control part for the simpleFoam is shown on the Listing 5.5.

```
application     simpleFoam;

startFrom       latestTime;

stopAt          endTime;

endTime         2000;

deltaT          1;
```

Listing 5.5: ControlDict time properties for simpleFoam.

The one corresponding to pisoFoam is shown on the Listing 5.6.

```
application     pisoFoam;

startFrom       latestTime;

stopAt          endTime;

endTime         10;

deltaT          0.00009;
```

Listing 5.6: ControlDict time properties for pisoFoam.

5.3.1.2 Data writing

The writing settings are composed by the frequency of writing solutions in the case folder, the writing format, the precision of this writing and the time precision.

The `writeControl` controls the timing of the writing solutions, if *timeStep* is the entry, the writing data is done every `writeInterval` time steps. For the *simpleFoam* solver the `writeInterval` has been assigned in a way that the solver writes only one output folder, as later this folder has to be deleted for the optimization to continue.

The `purgeWrite` is to delete and substitute solution folders, it has been set to 0 to turn off the option. The `writeFormat` has been chosen ASCII instead of binary, and for this reason a `writePrecision` has to be established, 6 by default. The `writeCompression` has been turned *on* in order to compress the written files. The `timeFormat` has been entered as *fixed* so the `timePrecision` does not change, 6 by default.

Finally, the `runTimeModifiable` has been set to *true* so the *controlDict* settings can be changed during a simulation, since enables re-read during a simulation at the beginning of each time step.

The resulting controlDic writing control is represented on the Listing 5.7.

```
writeControl    timeStep;

writeInterval   500;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression on;

timeFormat      fixed;

timePrecision   6;

runTimeModifiable true;
```

LISTING 5.7: ControlDict writing properties for simpleFoam.

5.3.1.3 Post-processing functions

Different functions can be added at the end of the *controlDict* in order to obtain the post-process data wanted.

First, with the aim to see the convergence of the solution, the residuals are computed with the function `#inlcudeFunc residuals`.

The force coefficients are the output needed for the optimization to compare the different flap positions. These coefficients are enabled with the function `type forceCoeffs` adding the library *libforces.so*. The writing controls have the same meaning explained before, so the force coefficients will be an output at every time step. It is important to set in which patches the coefficients will be calculated, these are the airfoil and Fowler flap.

In order to calculate the coefficients correctly, the lift and drag directions (`liftDir` and `dragDir`) have to be specified according to the wind direction, or what is the same, the angle of attack direction. Also, the density (`rhoInf`), velocity (`magUInf`), center of rotation (`CofR`) and pitch axis (`pitchAxis`) have to be defined. The reference length (`lRef`) is the chord length and the reference area (`Aref`) is the $lRef \times span$, where the span is the length of the z-direction extrusion.

```
functions
{
 #includeFunc residuals
 forces
    {
        type            forceCoeffs;
        libs ( "libforces.so" );
        writeControl   timeStep;
        writeInterval  1;
        patches         ("airfoil_intrados" "airfoil_extrados" "fowler_intrados"↩
             "fowler_extrados");
        rho             rhoInf;
        log             true;
        rhoInf          1.225;
        liftDir         (-0.069756473744125 0.997564050259824 0);
        dragDir         (0.997564050259824 0.069756473744125 0);
        CofR            (0.05 0 0);
        pitchAxis       (0 0 1);
        magUInf         5;
        lRef            0.11;
        Aref            0.00055;
    }
```

LISTING 5.8: ControlDict functions to calculate the residuals and the force coefficients.

Additionally, the y^+ function has been added since, as explained in section 3.3.1.1, the calculated dimensionless wall distance is not totally accurate due to the *flat plate approximation*. The real y^+ will be calculated and verified to be inferior than 5.

```
yPlus1
    {
        type         yPlus;
        libs         ("libfieldFunctionObjects.so");
        writeControl   timeStep;
        writeInterval  200000;
    }
```

LISTING 5.9: ControlDict function to calculate the y^+

5.3.2 fvSchemes

It is the directory where the numerical schemes for terms calculated during the simulation (time derivative, gradients, laplacian, etc) are specified. Since the case is solved as steady the time derivatives (`ddtSchemes`) are defined as *steadyState* which sets de derivatives to zero. In the lines below can be seen a piece of the file with the different schemes selected for each term, where the `gradSchemes` is for the gradients (∇), the `divSchemes` for the divergence ($\nabla\bullet$) and the `laplacianSchemes` for the laplacian (∇^2). A part of the file is shown on the Listing 5.10, the complete file can be found in the Appendix A.2.3.2.

```
ddtSchemes
{
    default          steadyState;
}
gradSchemes
{
    default          Gauss linear;
}
divSchemes
{
    default          Gauss linear;
    div(phi,U)       Gauss limitedLinearV 1;
    div(phi,R)       Gauss limitedLinear 1;
    div(phi,nuTilda) Gauss limitedLinear 1;
}
laplacianSchemes
{
    default          Gauss linear corrected;
}
```

LISTING 5.10: A piece of the fvSchemes file.

The `Gauss linear` schemes represent the Gaussian inegration with the standard finite volume discretisation with values interpolations from cell centres to face centres. The `limitedLinear` scheme limits regions of gradients changing highly. 1 is the strongest limit, tending towards linear with lower values until 0.

When there is only a default entry, it is not necessary to specify each term (pressure gradient, velocity gradient, etc). If any of these are specified, they dominate over the default. If *none* is entered at default, all the terms have to be specified.

The schemes have been selected following the OpenFOAM recommendations [13].

5.3.3 fvSolution

The *fvSolution* dictionary is where the equation solvers, tolerances and algorithms are placed.

The `solver` sub-dictionary specifies which linear-solver is used for the discretised equations. In the *fvSolution* file of the case, two linear-solvers are chosen:

- `GAMG`: generalised geometric-algebraic multi-grid.
- `smoothSolver`: solver that uses a smoother. These solvers require the smoother to be specified, in this case it is the `GaussSeidel` algorithm. When using the smooth solvers, the number of sweeps before the residual is recalculated can be specified, by `nSweeps`.

The `tolerance` is defined for the residuals, the solver stops when the residual reaches its value. The `relTol` is the relative tolerance, the ratio of current to initial residuals, the solver also stops when reaches its value. Also, a maximumum number of iterations can be defined in order to stop the solver when it arrives, `maxIter`.

```
solvers
{
    p
    {
        solver          GAMG;
        tolerance       1e-06;
        relTol          0;
        smoother        GaussSeidel;
    }
   ...
```

Listing 5.11: Linear-solvers specification in the fvSolution file

The loop which solves the governing equations with the SIMPLE algorithm is explained in the section 4.2, the parameters of this loop are chosen in the *fvSolution* directory in the sub-directory of the algorithm, in this case, `SIMPLE`.

The `nNonOrthogonalCorrectors` parameter, used by all algorithms, specifies repeated solutions of the pressure equation, used to update the explicit non-orthogonal correction of the Laplacian term, particularly set to 0 for steady-state. The `residualControl` specifies the residuals that have to be achieved when resolving the governing equations.

```
SIMPLE
{   nNonOrthogonalCorrectors 0;
    residualControl
    {
        p               1e-6;
        U               1e-6;
        nuTilda         1e-6;
    }
}
```

LISTING 5.12: SIMPLE algorithm properties in the fvSolution file

Finally, the `relaxationFactors` function controls the stability of the computations, avoiding under-relaxation, particularly used in steady-state. Under-relaxation works by limiting the amount which a variable changes from one iteration to the next. The under-relaxation factor α goes between 0 (solution does not change with iterations) and 1 (α increases, under-relaxation decreases).

The under-relaxation factors of the case are shown below, chosen based on OpenFOAM recommendations, the value has to be small enough to ensure stable computation but large enough to move the iterative process forward.

```
relaxationFactors
{
    fields
    {
        p               0.3;
    }
    equations
    {
        U               0.7;
        nuTilda         0.7;
    }
}
```

LISTING 5.13: Relaxation facotrs of the fvSolution file

5.3.4 decomposeParDict

The *decomposeParDict* directory is the one that enables the parallel run, the parallelisation is done decomposing the domain. The geometry and initial time fields are divided in processors in order to solve each one in a different core.

The `numberOfSubdomains` is the number of cores that will be used for the parallelisation, so the number of processors created will be the same.

The `method` entry refers to the method of decomposition used, the *simple* one means a geometric decomposition in which the domain is split into pieces by direction. Choosing the *simple* forces to specify the `simpleCoeffs` which are `n`, the number of sub-domains in each direction (x, y, z) and needs to multiply the number of cores. The `delta` is the cell skew factor, the objective of this parameter is to avoid using the faces of highly skewed cells being used as the processor boundaries.

```
numberOfSubdomains 8;

method          simple;

simpleCoeffs
{

    n               (4 2 1);
    delta           0.01;

}
```

LISTING 5.14: decomposeParDict file

In order to parallelise the simulation, the mesh and fields have to be first decomposed using the `decomposePar` utility.

Afterwards, the case must be executed with the `mpirun -np 8 simpleFoam -parallel` command, which needs the *mpi* modules uploaded. The execution needs the number of processors that will be parallelised (8) and which action is made in parallel (`simpleFoam`).

When the simulation is finished the solution is divided in the processors folders, the `reconstructPar` utility must be used to obtain the whole solution.

5.4 Automation of the solving

The case folders and files needed to solve each Fowler flap position are the ones explained in this section, and are the same for all geometry changes, these changes will only be reflected in the *polyMesh*.

It is important to note that, when using *simpleFoam* as solver, which does not have time integration schemes, the variation of the mesh does not influence the election of the time step, however, with *pisoFoam* the convergence could fail for some meshes and not for others, so in case *pisoFoam* is implemented in the future, the time step would have to be chosen as the lowest encountered.

All the case files and folders need to be ready with the new *polyMesh* in order to solve the new flap position case. It is essential that, when a case is finished and before the next starts, all the solution folders are deleted, if not the solving will start from that solution time (*latestTime*) as it is chosen in the *controlDict* directory, and would not solve the case as the folder time will be already the *endTime*. Additionally, the *polyMesh* folder and the *.unv* mesh have to be deleted in case the mesh of the new flap position fails and does not subscribe the new *polyMesh*, in that case, the solver would solve again the previous mesh and the optimizer would assign the results to the new flap position when instead were from the previous one.

In order to parallelise the run of each iteration, the command `decomposePar` will have to be executed every time before running in order to decompose the domain. The following step is to call `mpirun -np 8 simpleFoam -parallel` to run the simulation in the established number of cores. When it is finished, it is not needed to `reconstructPar` because the only values that the optimizer needs are the ones saved in the forces folder, which will not be decomposed. When the solution of the case is deleted, it is done directly from the decomposed processors folders.

The script executing all the actions mentioned will be explained and exposed in the coupling section 6.2.2.

Chapter 6

Optimization

The current Chapter presents the set up of the optimization with RMOP2.0 and the coupling with the automated pre-process and solving explained in the previous chapters. Both have been accomplished thanks to the easy and practical usage of the RMOP2.0 platform.

In order to accomplish the optimization, the RMOP2.0 required parameters have to be selected for the case. Afterwards, the *Analyser* script, which is in charge of giving the calculated coefficients for each flap position, will be described. The coupling process will be represented in a flow diagram.

Finally, the files and executables that are used during the Analyser process will be described in order to understand the functioning of the entire optimization process.

6.1 Optimization parameters

In this Section a more intensive description is made along with the specification of the chosen parameters for the present optimization.

6.1.1 Objective functions specification

As RMOP2.0 is a multi-objective optimization platform, more than one objective function can be established to optimize. In the present optimization case the objective functions will be the lift coefficient (C_l) and the aerodynamic efficiency ($E = \frac{C_l}{C_d}$) of the airfoil and flap. In this way, the maximum lift coefficients will be obtained for the best efficiency coefficients, and vice versa. As RMOP2.0 is configured to minimise the objective functions, the sign of E and C_l has been changed, causing the minimal solution to be the most wanted.

6.1.2 Design variables specification

The design variables that will be changed at every iteration are the flap deflection angle δ and the positions dx and dy. As the design variables are positions proposed by RMOP2.0, it is needed to specify in which range these variables can be chosen and the step of the values from the lower to the upper bounds, if these variables are discrete.

The bounds have been chosen due to the limits of the mesh, as explained in Chapter 3. As the dy limits are dependant of the angle, the range of dy movements will be [-1,1] and interpolations will be made within the calculated limits of each range of angles. This is implemented in the file that reads the design variables explained in the Section 6.2.3.2. The steps and the limits for each design variable are represented on the Table 6.1 on the Section 6.2.3.2.

Optimization formulation

The optimization case can be formulated as

$$Cl : \chi \to R \quad \min(-Cl(x,y,\delta)) \qquad E : \chi \to R \quad \min(-E(x,y,\delta)) \qquad \begin{matrix} \mathrm{x} \in \chi_1 \\ \mathrm{y} \in \chi_2 \\ \delta \in \chi_3 \end{matrix}$$

where $\chi_1 = [-0.003, 0.003]$, $\chi_2 = [-1, 1]$, $\chi_3 = [0, 30]$

6.1.3 Initial population

The process begins with a set of individuals which is called a population. The population size is static so the new generations replace the old ones. It is important for the problem to evolve correctly the maintenance of the population diversity, otherwise it might lead to premature convergence. The size of the population should be determined correctly in order to avoid slow down the process or having insufficient population for a good solution. The population for the case has been chosen of 16 individuals, as suggested by RMOP developers this value has to be more than 4 times the number of the design variables and multiple of the number of CPUs. The population will be initialised as random.

6.1.4 Selection

Pairs of individuals are selected based on their scores in the objective function. Individuals with high score values have more chance to be selected for reproduction. In RMOP2.0, the selection is based in non-dominated sort of the $\mu + \lambda$ population, where μ is the size of the parent population and λ is the size of the offspring. The selection process works starting with the initialisation of the population size μ, and λ species are selected from the population using the Tournament selection where a random group of individuals are taken from the population, and the most fitted are selected as parents.

6.1.5 Recombination

The recombination is the process by which selected individuals are recombined to form a offspring. There are two main components, the operators of crossover and mutation.

Crossover

The crossover operator represents the mixing of parents to create the offspring. In RMOP2.0 the crossover operator used by default is the Simulated Binary Crossover (SBX), which uses two parent vectors and combines them to create two new individuals. After crossover, the resultant individuals will be passed on to the mutation stage.

Mutation

In order to maintain diversity and avoid the algorithm to block, a mutation on some of the children produced is carried out, varying randomly some of their properties. In RMOP2.0 the mutation operator is the Polynomial Mutation, where a polynomial probability distribution is used to perturb the solution obtained by the crossover.

6.2 Coupling the optimization

Having the automated process of geometry, domain, mesh and solution for the different flap changes, it is necessary to couple the process with RMOP2.0. The optimizer makes the coupling an easy process by three actions; it writes the design variables in the *Eval.DVs* file, it executes the *analyser* and waits until the process is finished to finally read the results placed in the *Eval.individual* file. Thus, the solving process must function in an automated way interacting with these two files.

First, the RMOP2.0 entries will be set up.

6.2.1 RMOP2.0 input

The input file for RMOP2.0 (*input.RMOP*) is composed by the following components. The code lines can be found in the Appendix A.3.1.

- **Number of Design Variables**, which, as explained on the previous section, are three: dx and dy flap movements and the angle deflection (δ).
- **Number of Objectives functions**, two have been determined, the negative values of efficiency -E and lift coefficient -C_l.
- **Analyser File**, the main script that will read the *Eval.DVs* with the flap positions, run the process and write the *Eval.individual* file with the results (E and C_l).
- **Number and name of essential files**, the files and folders that will be called and managed by the analyser file. These are explained in the Section 6.2.3.
- **Population size**, number of individuals of each generation, chosen of 16.
- **Random Initial Population**, number of initial random design variables.
- **Probability of crossover and of mutation**, probability that an individual is selected to apply the crossover or mutation operator to it.
- **Lower and upper bounds** of the design variables.
- **Variable step**, the step for choosing the values between the lower and upper bounds of the design variables if they are discrete values.
- **Termination criteria**, which has been specified by 300 evaluations. Higher evaluations would have been chosen if the available time for the run was higher.

6.2.2 Analyser

The analyser file is the responsible of, given a new flap position parameters, attach them the objective functions values. This is accomplished following the procedure explained in the sections *Preparation of the case* 3 and *Solving the case* 4 in an automated way.

This automated process is carried out thanks to the main script *ANALYSER.sh* and to the files that it uses to function.

This analyser file is shown on the Listing below and can also be found in the Appendix A.4.1, and its functioning process will be explained next.

```
#!/bin/bash

# in case the individual fails
python DATA0.py

printf "Optimization changes"
python GEOMmodification.py

printf "S A L O M E"
printf "Creating the mesh around the new flap position\n\n"
salome POINTS.py MESH.py

printf "O p e n F O A M"
# Transform Salome mesh to OpenFOAM
printf "\n\n Changing the Salome mesh to OpenFOAM \n\n"
cd OpenFOAM/
ideasUnvToFoam Compound_Mesh.unv
cd ..
# replace the boundary conditions with the ones wanted
cp boundary OpenFOAM/constant/polyMesh/
printf "\n\n CHECK MESH \n\n"
cd OpenFOAM/
checkMesh
printf "\n\n SOLVING THE CASE WITH SIMPLEFOAM \n\n"
decomposePar
mpirun -np 8 simpleFoam -parallel
cd ..

printf "Resultant data"
python DATA.py

# restart the case
rm -r OpenFOAM/postProcessing OpenFOAM/processor* OpenFOAM/constant/polyMesh ↩
    OpenFOAM/Compound_Mesh.unv
```

LISTING 6.1: Analyser file, *ANALYSER.sh*

The first thing that is done when the process begins is to establish bad values for the objective functions as, in the case the process breaks at some point, there is no possibility that good values from previous evaluations will be used. After, a Python script is executed (GEOMmodification.py) that reads the *Eval.DVs* file and assigns the flap variations to another Python file (POINTS.py), which will be later executed by Salome 9.2.1 together with the file that creates the mesh (MESH.py). The successive step is to convert the mesh to OpenFOAM, replace the boundary file and check the mesh quality. At this point, the parallelised solving of the case can be run. Finally, the resultant values of the CFD solution are placed in the *Eval.individual* file by the DATA.py Python file, so the optimizer can evaluate them. At the end of the process, the OpenFOAM results are removed from the case to avoid them disturb the next iteration, as, in case of failure, the results would be taken by DATA.py.

The complete process is represented in the flow diagram on the Figure 6.1.

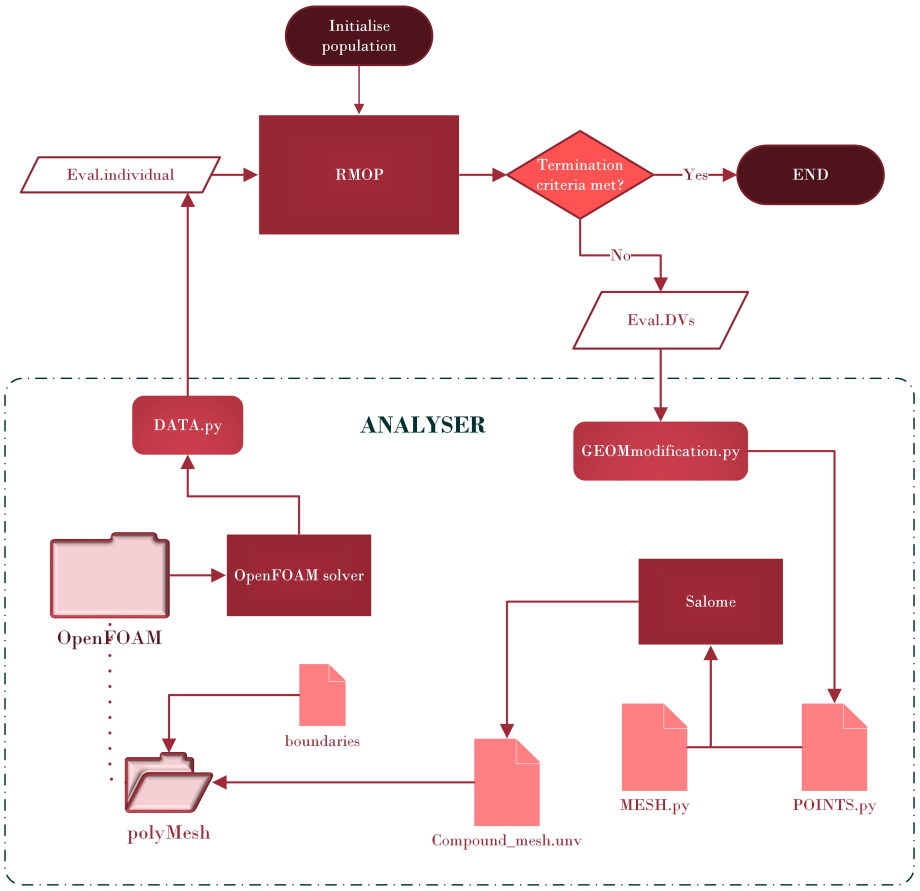


FIGURE 6.1: Flow diagram of the optimization process

On the following section a more deep description will take place of the role of each essential file and its execution code.

6.2.3 Essential files

The essential files or folders (with all its files) are the ones that have to be imported in order to perform the optimization. Not all the files used during the process are considered essential, the ones that are created by other files on the process do not need to be an input of the optimization process.

- **Essential files and folders:** DATA0.py, GEOMmodification.py, MESH.py, OpenFOAM folder, boundary, DATA.py
- **Not essential files:** POINTS.py, OpenFOAM/postProcessing, OpenFOAM/processors, OpenFOAM/constant/polyMesh, OpenFOAM/Compound_Mesh.unv

6.2.3.1 Resetting the objective functions (DATA0.py)

All detected failures of the process have been corrected, however, there is still the possibility that some individual goes wrong. If this happens, it is important that the optimizer does not consider that individual as a good one, and that the same individual is not proposed again. In the case the process goes wrong the optimiser could get the values of the previous iteration or some others, as no values would be defined in the *Eval.individual* file. To prevent this to become a problem, the values on the *Eval.individual* file will be established from the beginning as 0. This way the optimizer will not want to produce new generations of these positions.

This is done by the Python file DATA0.py which is composed by a simple code that can be found in the Appendix A.4.2.

6.2.3.2 Reading the design variables (GEOMmodification.py)

The GEOMmodification.py file has the airfoil and flap points imported, which could be read from an external file, but this would increase the number of essential files, so its definition has been made directly on the script.

The *Eval.DVs* file is read by the GEOMmodification script and the values of dx, dy and δ are assigned to the variables following the constraints delimited by the mesh.

The bounds delimitation is exposed in the *Mesh limitations* Section 3.3.5. As explained previously, it is used an interpolation between the limitation values of the design variable dy_DV and the limits of dy in function of the deflection angle δ.

The limits of the design variables have been defined as represented on the Table 6.1.

	dx	dy_{DV}	δ
Variable step	0.0005	0.2	0.5
Lower bounds	-0.003	-1	0
Upper bounds	0.003	1	30

TABLE 6.1: Design variables bounds and steps.

The interpolation made is the one represented on the equation (6.1).

$$\frac{dy_{DV}+1}{1-1} = \frac{dy - bound_{low}}{bound_{up} - bound_{low}} \tag{6.1}$$

The limits of dy for each range of angles are represented on the Table 6.2.

	dy upper limit	**dy lower limit**
0<delta<5º	0.004	-0.003
5º<delta<10º	0.002	-0.004
10º<delta<15º	0.0015	-0.003
15º<delta<20º	0.001	-0.0025
20º<delta<25º	0.0005	-0.002
25º<delta<30º	0.0005	-0.003

TABLE 6.2: dy upper and lower bounds in function of the deflection angle δ.

An part of the code implementing these interpolations is shown in the Listing 6.2.

```
with open( Eval.DVs ,  r ) as DV:              # import dx and dy
      linesplit = line.strip().split()
         dx = float(linesplit[0])
            angle = float(linesplit[2])
      if angle <= 5:
         dy = (float(linesplit[1])+1)*(0.004+0.003)/2-0.003
      if 5 < angle <= 10:
         dy = (float(linesplit[1])+1)*(0.002+0.004)/2-0.004
```

LISTING 6.2: GEOMmodification.py dy constraints

When the variables are assigned, the POINTS.py file is created with the changed position of the flap. This process has been explained in the *Geometry automation* Section 3.1.1.

6.2.3.3 Creating the geometry, domain and mesh (MESH.py)

The MESH.py file contains all the commands to create the geometry, domain and mesh around the new points of the POINTS.py file. The file can be found in the Appendix A.1.3.

When both the POINTS.py and MESH.py files are opened with Salome, the whole process described in Chapter 3 is executed, and the mesh is saved as Compound_Mesh.unv in the OpenFOAM folder.

6.2.3.4 Solving with OpenFOAM (OpenFOAM folder and boundary file)

The essential OpenFOAM folder contains all the files described on Chapter 4 except the ones produced by the mesh (PolyMesh) or created when solving the case.

In the section 3.3.3 it is explained the procedure followed from the Salome mesh (Compound_Mesh.unv) to the polyMesh. As it is also explained, the boundary file must be changed to correctly assign the patches, this is why the boundary file is also an essential file, as it has to replace the one created inside the polyMesh folder. Consequently, having the OpenFOAM folder complete, the case can now be run and the solution created.

6.2.3.5 Saving the resultant values (DATA.py)

The solution of the case creates the solved results post-process data inside the *postProcessing* folder. The file that has the values for the optimizer to evaluate this flap position is the forceCoeffs file inside the *forces* folder.

The file is opened and read by the DATA.py script. In the case of simpleFoam the aerodynamic coefficient values can be obtained as the last value calculated, as it does not compute changes in time, thus, it does not capture the oscillation of the forces. On the contrary, if pisoFoam is used, since it does capture oscillations the values must be obtained making an average of the converged results. The oscillation mentioned is represented on the Figure 6.2.

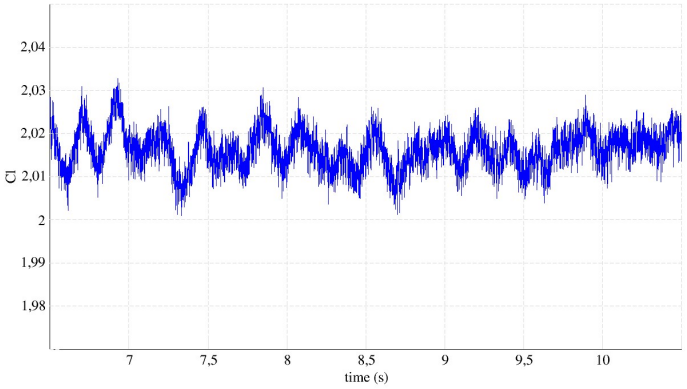


FIGURE 6.2: Cl oscillation for δ=22º with pisoFoam

In this case, it is necessary to discard the first seconds of the results, and calculate the mean value of the converged solution. Using pisoFoam the first 45000 lines of the *forceCoeffs* file have to be omitted, 4 seconds of results approximately. With simpleFoam are 1500. The unstable part of the solution can be observed for each solver in the Figure 5.1, Chapter 5.

From the values of C_l and C_d the efficiency is obtained, and both solution values C_l and E are set negative as RMOP2.0 optimizes through minimization.

In order to be able to implement pisoFoam in future optimizations, the script has been set with the average coefficients calculation. The script can be seen in the Listing 6.3 and also can be found in the Appendix A.4.3.

```
with open( OpenFOAM/postProcessing/forces/0/forceCoeffs.dat ,  r ) as OFV:
   for line_number, line in enumerate(OFV, 1):
         if line_number <= not_wanted_lines:
                  continue
         clvalues.append(float(linesplit[3]))
         cdvalues.append(float(linesplit[2]))
   cl = sum(clvalues)/float(len(clvalues))
   cd = sum(cdvalues)/float(len(cdvalues))
   E = cl/cd
OFV.close
with open( Eval.individual , wa ) as IND:
   IND.write(str(cl)+   +str(E))
IND.close
```

LISTING 6.3: DATA.py saving the objective functions

After the DATA.py execution, RMOP2.0 already has the *Eval.individual* file corresponding to the *Eval.DVs* values, and can proceed to evaluate the next individual.

6.3 Summary

The RMOP2.0 optimization parameters have been defined and specified for the current optimization case. These are the objective functions, the design variables, the initial population and the process of selection, recombination, crossover and mutation.

Afterwards, the RMOP2.0 configuration file has been prepared and its inputs and outputs needed for the coupling have been described.

Consequently, knowing how the coupling will be accomplished, the file to carry out the coupling is introduced. This file is the *Analyser* file and it contains all the automated process independent to RMOP2.0. The *Analyser* is the responsible for calling all the sub-processes in order to give the objective functions for a determined design variables.

Then, the necessary files for these sub-processes have been exposed and their tasks explained, completing the description of how the entire optimization process works.

Chapter 7

Optimization results

The optimization process can be finally run as the coupling is now accomplished and all the sub-processes are automated.

The execution of the optimization will be the validation that the coupling works and that the geometry, domain, mesh and solver do work in an automated way finding better results for some determinate positions.

After a brief introduction about the run properties and the computer resources used, the results obtained by RMOP2.0 are evaluated and discussed.

7.1 Results conditions

The optimization process has been run for a population of 16 individuals, with a random initial population of 28 individuals. The number of evaluations has been 300, set as the termination criteria, thus, the number of the generations computed has been 19.

The optimization has finished when the termination criteria has been met, with an elapsed time of 6 days, 11 hours, 45 minutes and 30 seconds. The computation resource used for running the process has been an Intel Core i7 8th generation with 8 cores provided by CIMNE. The CFD analysis has been parallelised for each individual reducing significantly the computational time.

7.2 Convergence

The number of evaluations used can be considered acceptable as the values of lift coefficient and efficiency seem to be converged, which can be seen on the Figures 7.1 and 7.2. The convergence is evaluated as the better solutions of Cl and E encountered throughout the evaluations. The optimization is converged when no better solutions are found.

On the Figure 7.1 the convergence evolution for the first objective function, Cl, is shown along the evaluations.

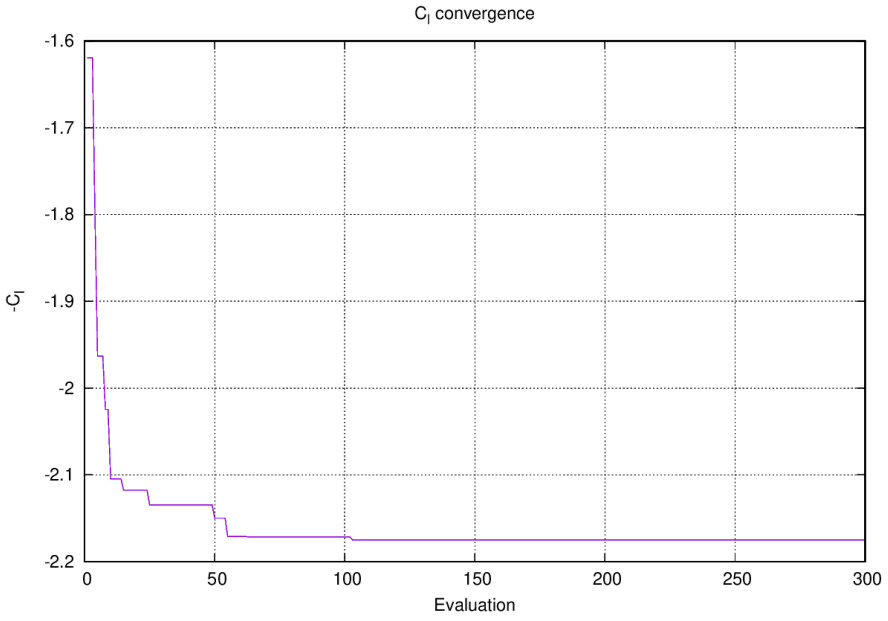


FIGURE 7.1: Cl convergence over evaluations.

On the Figure 7.2 the convergence evolution for the second objective function, E, is displayed along the evaluations.

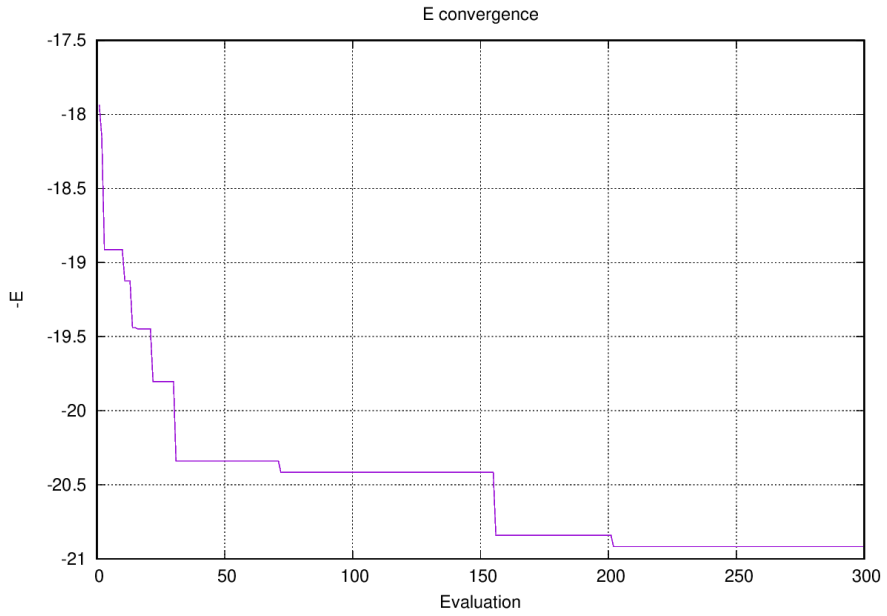


FIGURE 7.2: E convergence over evaluations.

It can be observed that the Cl values have converged with fewer evaluations than the E values. Both objective functions have rapidly improved their best results on the first evaluations, and later the improvements have been attenuated until reaching the maximum value encountered.

A comparative table has been made between the two objective functions, Table 7.1, where the increase of the best values from the first evaluation to the last is represented. Also the evaluation at which the maximum values have been reached is shown.

	Cl	**E**
First value	1.6196	17.9365
Maximum value	2.1751	20.9176
Increase (%)	34.3	16.6
Evaluations for the maximum value	103	202

TABLE 7.1: Comparative table between the objective functions convergence.

The number of evaluations could be increased in order to reassure the convergence or, in the case it is not converged, find new better values.

7.3 Pareto Front

Having a multi-objective optimization with two objective functions, the resulting Pareto, composed by the non-dominated values of Cl and E, can be plotted in a 2D graph. This way, the best configurations of the flap can be easily seen having into account both the lift coefficient and the efficiency.

The Pareto Front obtained in the optimization is shown on the Figure 7.3.

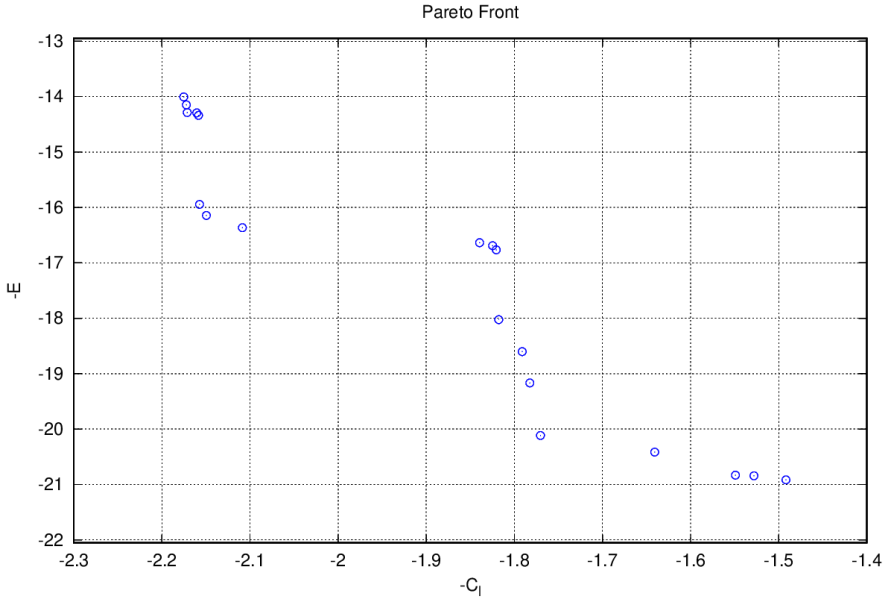


FIGURE 7.3: Pareto Front.

The values composing the Pareto Front are considered equally good solutions, as can not be sorted from better to worse in an objective way. The selection process inside the Pareto has to be done by subjective preferences such as preferred higher efficiency in exchange of lower lift coefficient or vice versa.

A higher number of Pareto points could be obtained reducing the step values between the maximum and minimum limits of the design variables.

The values conforming the final Pareto Front are defined in the Table 7.2 with its respective design variables. As can be seen, the dx values are only moved in the region between $dx = 0$ and $dx = 0.001$, a lower step should be used in order to achieve more accurate positions in this range. As expected, the higher values of Cl are achieved with higher deflection angles δ while higher efficency is encountered with lower δ. The dy values of the Pareto Front show that better results are obtained when the flap is near the airfoil, but these positions depend on the dx and δ values.

Cl	E	dx	dy	δ	Cl	E	dx	dy	δ
2.175	14.005	5×10^{-4}	0.8	29.5	1.821	16.765	0.0	0.8	17.0
2.172	14.148	1×10^{-3}	0.6	29.5	1.818	18.024	5×10^{-4}	0.8	15.5
2.171	14.286	1×10^{-3}	0.4	29.0	1.791	18.606	5×10^{-4}	1.0	14.0
2.160	14.294	1×10^{-3}	0.2	29.0	1.782	19.166	5×10^{-4}	1.0	13.0
2.158	14.339	1×10^{-3}	0.6	29.0	1.770	20.115	5×10^{-4}	1.0	11.5
2.157	15.945	1×10^{-3}	0.8	25.5	1.641	20.415	5×10^{-4}	1.0	8.0
2.149	16.144	1×10^{-3}	0.8	25.0	1.549	20.830	0.0	1.0	5.5
2.108	16.366	5×10^{-4}	0.8	24.5	1.528	20.841	0.0	1.0	5.0
1.839	16.635	5×10^{-4}	0.8	17.5	1.492	20.917	5×10^{-4}	1.0	4.0
1.825	16.690	5×10^{-4}	0.6	17.0					

TABLE 7.2: Pareto Cl and E values, and its corresponding dx, dy and δ.

The Pareto front is computed at the end of each generation, finding the individuals that are not dominated by the rest of individuals. The Pareto evolution through the generations is represented on the Figure 7.4.

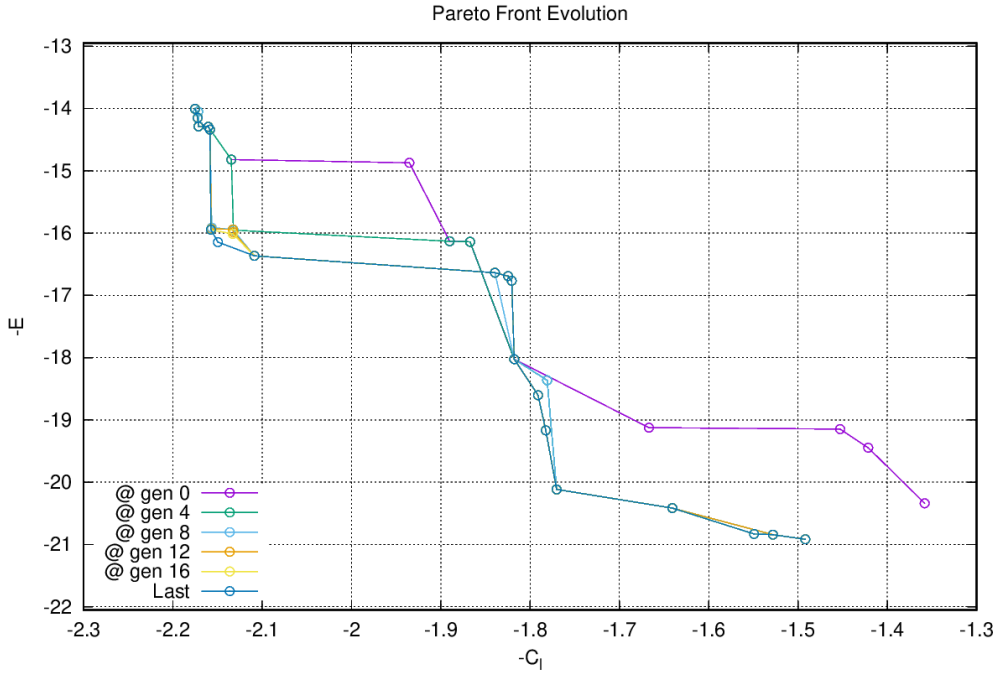


FIGURE 7.4: Pareto Front evolution.

The regions where higher improvements have been found through generations are in high efficiency values with low lift coefficient and the other way round. The central region has not experienced large evolution along generations.

7.4 Initial buffer, parents and children

The initial random buffer is plotted with the final Pareto in order to see the improvements accomplished from the beginning to the end. This is shown on the Figure 7.5.

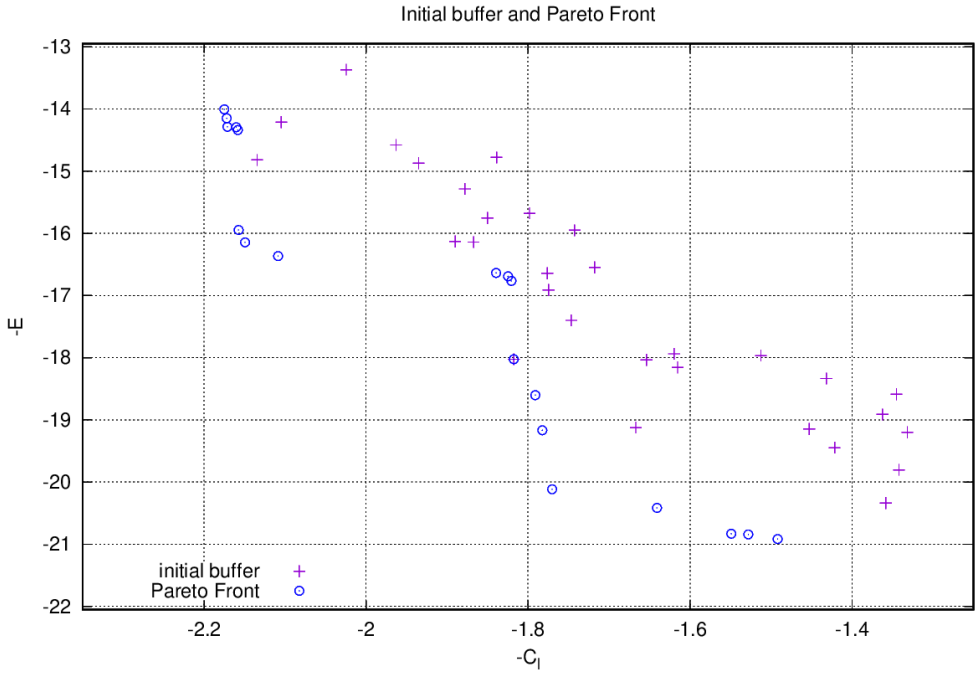


FIGURE 7.5: Initial buffer and final Pareto Front.

As can be seen on the Figure 7.5, the distances between the buffer and the Pareto Front at the centre values are much shorter than the ones on the extremes. Even one position found by the random buffer is part of the final Pareto. On the other hand, the optimized values obtained for the maximum regions of Cl and E have increased considerably its results.

An interesting result of the optimization is the increase of Cl in the efficiency values. For example, the highest E found in the initial buffer has been improved only a 2.85% but, at the same time, a 9.87% of its Cl has been enhanced. The same happens for low values of E where, for the same values of E, the improvements in Cl have been around the 20%.

In order to have higher improvements on the objective functions the limits of the design variables could be extended, however, it is possible that the best positions of the flap have already been found and that the highest improvements of the flap position are around the percentages mentioned. What definitely should be done is decrease the step values of the design variables in order to increase the number of the Pareto points.

In order to confirm the evolution of the results, the objective functions of the initial buffer are plotted with the first parents. This is displayed on the Figure 7.6.

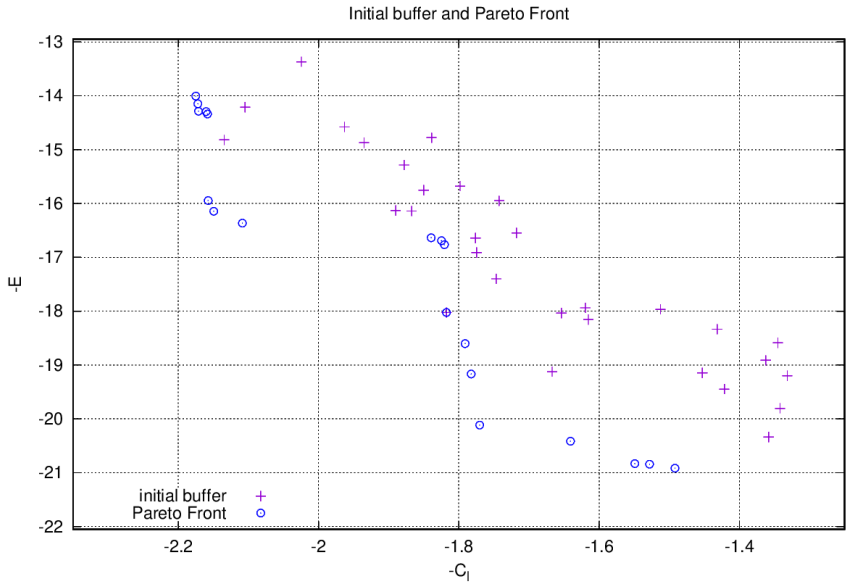


FIGURE 7.6: Buffer and first parents.

As expected, the first parents are the ones with the best solutions in both Cl and E.

Moreover, the parents and children of the optimization are displayed on the Figure 7.7 and it can be noted that the parents are placed near the best regions while children are more disperse. This is because parents are selected with the best results while children are submitted to crossover and mutation, not always finding better results than before.

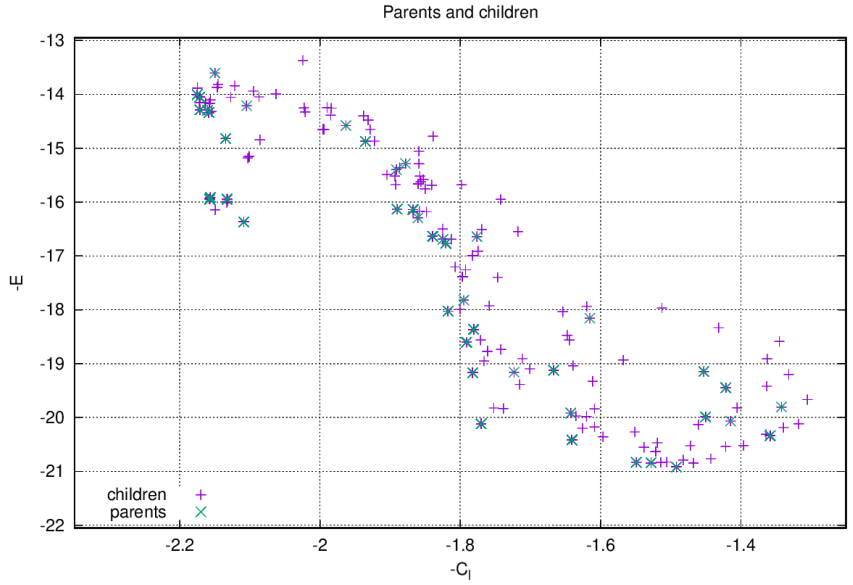
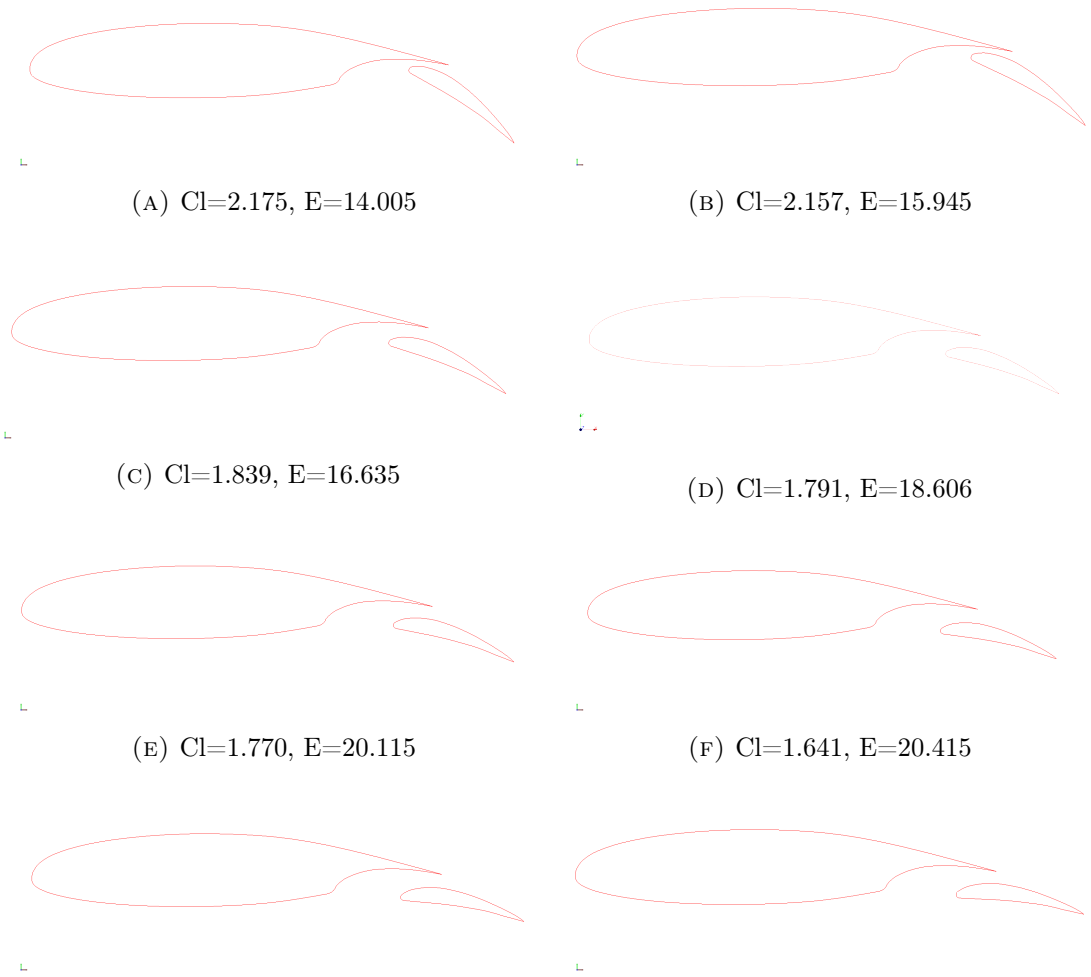


FIGURE 7.7: Parents and children.

7.5 Optimized flap positions

The optimization process has been run satisfactorily. Some of the best design variables encountered for the objective functions set are the ones represented in the Figure 7.8. The rest have not been displayed as they have similar positions, their positions and objective function values can be found in the Table 7.2.



(A) Cl=2.175, E=14.005

(B) Cl=2.157, E=15.945

(C) Cl=1.839, E=16.635

(D) Cl=1.791, E=18.606

(E) Cl=1.770, E=20.115

(F) Cl=1.641, E=20.415

(G) Cl=1.549, E=20.830

(H) Cl=1.492, E=20.917

FIGURE 7.8: Some of the Pareto Front flap positions with their respective objective functions obtained.

As previously commented with the Table 7.2, these results have sense in terms of aerodynamics, however, it is mandatory to have in mind that the process has been made with approximations in order to get an acceptable solution within the available time. For this reason, a more accurate process should be made in order to confirm that these are the best flap positions for optimum Cl and E, and that these values of Cl and E are correct.

Chapter 8

Conclusions and future work

The work presented in this Thesis has fulfilled the main goal of the project, the automated coupling between the optimizer RMOP2.0, the meshing software Salome and the CFD software OpenFOAM to optimize aerodynamic geometries. Even though the process developed has aspects to be improved, the performance achieved has been satisfactory and can be a proper start for further development.

It has been able to prepare an entire CFD case carefully developed even though the selected parameters have not been the most accurate ones. This has been due to the search for the minimum computational time, requirement that comes from the large number of evaluations that the optimization needs to compute in order to see a more or less converged solution.

It has been able to automate a sufficient good mesh that captures the flow behaviour around the geometry, the mesh is created around the points that conform the geometry and changes along with them, also the boundary conditions have been successfully automated. Furthermore, with the same automated process more accurate results can be prepared with a finer mesh or with a solver that considers more phenomenons, it is therefore possible to increment the accuracy of the case using the same automated process. It should be point out that the process developed for the creation of the mesh it is not only valid for the tested case, other airfoils could also follow the same automated process.

Different data transference have had to be done, not only between the optimizer and the analyser but also in the sub-processes carried out in the analyser, such as the mesh conversion or the points transference. All these have been successfully achieved with a reduced number of files.

The optimization solution has resulted in a coherent Pareto of objective functions and its corresponding design variables. Nevertheless, the optimization process should be run in other conditions to assure the values obtained, these are explained in the explanation of the future work.

As a highly developed optimizer has been used for the optimization process, a large amount of knowledge has been acquired in the optimization matters with multi-objective functions and genetic algorithms.

It is important to remark that this Thesis has been made in collaboration with the recognised research centre CIMNE (International Centre for Numerical Methods in Engineering) providing the use of the optimizer RMOP2.0 and having a receiving a feedback of quality in the optimzation matters. This also has provided the opportunity to use high computational resources, accelerating the computation time of the simulations and acquiring knowledge related to the cluster.

It should be pointed out that the software used for the analyser process is open source (Salome and OpenFOAM), this implies not client-prepared services and thus, higher effort on using these platforms. However, in exchange it has been obtained a coupled process with high recognised platforms and with the minimum cost of software. Also these platforms, mainly OpenFOAM, allow the user to deeply understand the functioning of the software.

It should be mentioned the high number of obstacles that have appeared during the execution of the Thesis, mostly in the domain and mesh automation. These have been an inconvenience during the accomplishment of the objective, but also have forced the search and evaluation of different solutions, making to not stop working on the final purpose. Furthermore, it should be pointed out that, being a field currently in development, uncertainly of achievement of automation has appeared in determined stages of the project.

Finally, highlight that the time needed for the current Thesis has been perceived much larger that the provided for doing the Final Thesis, for this reason numerous suppositions have had to be made. Therefore, the current Thesis represent the first steps before a high quality aerodynamic optimization can be achieved.

The improvements that are suggested will be presented in the future work.

8.1 Future work

As mentioned several times along the Thesis, most of the parameters selected for the resolution of the case have been chosen directly influenced by the computational time. Consequently, in the future work of the study, as computational time will not suppose that much of a determinant factor, more accuracy on the result will be possible but with the same optimization process.

The first further analysis that should be made on the study is the comparison of the Pareto solution of the current optimization with the Pareto solution of the same case with more accurate parameters of resolution. These parameters are:

- Reduction of the dimensionless wall distance (y^+) of the mesh. Reducing this parameter to 1 or lower would increase the accuracy of the results in the boundary layer, and thus, the calculation of the aerodynamic coefficients. Even a mesh independence test should be made in order to choose a mesh that does not influence the results. In this case, carefully attention should be placed on the automation of the mesh for not degrading its quality.

- Transient effects taken into account in the calculation by choosing a different OpenFOAM solver. Even though the satisfactory results obtained in the comparative study that has been made on the Chapter 6 between a steady and a transient solvers, considering the transient effects of the fluid flow could change a little the Pareto front due to the vortexes.

- Higher evaluations on the optimization, in order to assure the convergence of the Pareto solution or find better solutions. More discretization in the design variables to obtain more Pareto points defining the optimum situations. Also a higher range of flap movements could be implemented if the mesh was made again. Defining it with lower restrictions near the optimized positions.

In a further future work, the same optimization process could be applied to different types of geometries as long as the mesh was made again.

Concluding, the field of aerodynamic optimization with CFD is an emerging field which has been the main motivation since the beginning of the study. Even though the complications encountered, the development of the study has been enjoyable and a lot of knowledge has been acquired in many aspects. Moreover, finally having achieved the main aim of the study has been incredibly satisfactory.

Chapter 9

Bibliography

[1] "Pareto front Definition in Multi-Objective optimizations."

[2] Katz, J., *Introduction to Computational Fluid Dynamics*, vol. M. 2012.

[3] for Numerical Methods in Engineering (CIMNE), I. C., "Official web page of Robust Multi-Objective Optimization Platform (RMOP)." http://tts.cimne.com/RMOP.

[4] Emmerich, M. T. M. and Deutz, A. H., "A tutorial on multiobjective optimization: fundamentals and evolutionary methods," *Natural Computing*, vol. 17, pp. 585–609, Sep 2018.

[5] Kenway, G. K. W., Martins, J. R. R. A., and Kennedy, G. J., "Aerostructural optimization of the Common Research Model configuration," tech. rep.

[6] Peigin, S. and Epstein, B., "Multiconstrained aerodynamic design of business jet by CFD driven optimization tool," *Aerospace Science and Technology*, vol. 12, pp. 125–134, mar 2008.

[7] HAFTKA, R. T., "Optimization of flexible wing structures subject to strength and induced drag constraints," *AIAA Journal*, vol. 15, pp. 1101–1106, aug 1977.

[8] Zhang, Z. J., Khosravi, S., and Zingg, D. W., "High-Fidelity Aerostructural Optimization with Integrated Geometry Parameterization and Mesh Movement," tech. rep.

[9] Piperni, P., Abdo, M., Kafyeke, F., and Isikveren, A., "Preliminary aerostructural optimization of a large business jet," *Journal of Aircraft - J AIRCRAFT*, vol. 44, pp. 1422–1438, 09 2007.

[10] Anderson Jr, J. D., *Fundamentals of aerodynamics*. Tata McGraw-Hill Education, 1985.

[11] "Simscale website, y+ definition." https://www.simscale.com/forum/t/what-is-y-yplus/82394.

[12] "OpenFOAM v6 User Guide: 5.1 Mesh description."

[13] "The open source CFD toolbox." https://www.openfoam.com/documentation/user-guide/standard-solvers.php.

[14] Modelling, N. T., "The spalart-allmaras turbulence model." https://turbmodels.larc.nasa.gov/spalart.html, 2019.

[15] Schlichting, H. T. and Truckenbrodt, E. A., *Aerodynamics of the Airplane*. McGraw-Hill Companies, 1979.

[16] for Numerical Methods in Engineering (CIMNE), I. C., "Tutorial 1 of RMOP." http://tts.cimne.com/RMOP/Docs/tutorials/T1_RMOP-XFOIL_ES.zip.

[17] "Menter, F. R., "Two-Equation Eddy-Viscosity Turbulence Models for Engineering Applications," AIAA Journal, Vol. 32, No. 8, August 1994, pp. 1598-1605.,"

[18] "Turbulence free-stream boundary conditions,"

[19] "Runge-Kutta method - Encyclopedia of Mathematics."

[20] Mathematics, B. P., "A computational fluid dynamics analysis on air bearing designs," no. June, 2015.

[21] "What is the difference between pressure based solver and density based solver?."

[22] "The open source CFD toolbox."

[23] Lin, C.-T., "New High-Resolution Central-Upwind Schemes for Nonlinear Hyperbolic Conservation Laws," tech. rep.

[24] Rahman, U. A. and Mustapha, F., "Validations Of Openfoam ® Steady State Compressible Solver Rhosimplefoam," *International Conference on Mechanical And Industrial Engineering*, p. 6, 2015.

[25] Fürst, J., "On the implicit density based OpenFOAM solver for turbulent compressible flows," *EPJ Web of Conferences*, vol. 143, no. 2, p. 02027, 2017.

[26] Chun, S., Fengxian, S., and Xinlin, X., "Analysis on capabilities of density-based solvers within OpenFOAM to distinguish aerothermal variables in diffusion boundary layer," *Chinese Journal of Aeronautics*, vol. 26, no. 6, pp. 1370–1379, 2013.